Títol: Desenvolupament d'una interfície web per a una aplicació per a la gestió de preguntes (LearnSQL Authoring Tool)

Volum: 1

Alumne: Albert Boada Flaquer

Director: Carme Quer Bosor

Departament: Enginyeria de Serveis i Sistemes d'Informació

Data: 26/09/2016

DADES DEL PROJECTE

Títol: Desenvolupament d'una interfície web per a una aplicació per a la gestió de preguntes (LearnSQL Authoring Tool

Nom de l'estudiant: ALBERT BOADA FLAQUER

Titulació: Enginyeria Informàtica

Crèdits: 37,5

Director/Ponent: Carme Quer Bosor

Departament: Enginyeria de Serveis i Sistemes d'Informació

MEMBRES DEL TRIBUNAL (Nom i signatura)

President: ***ALBERTO ABELLO GAMAZO***
Dept. Enginyeria de Serveis i Sistemes d'Informació

Vocal: ***RAMON CANAL CORRETGER***
Dept. Arquitectura de Computadors

Secretari: ***CARME QUER BOSOR***
Dept. Enginyeria de Serveis i Sistemes d'Informació

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Resum

Aquest projecte parteix de la ja consolidada eina d'escriptori *LearnSQL Authoring Tool* del departament d'*Enginyeria de Serveis i Sistemes (ESSI)* de la *Facultat d'Informàtica de Barcelona (FIB)* i l'evoluciona portant-la cap a la plataforma web.

L'eina és només una petita part d'un sistema més gran anomenat *LearnSQL*, el qual neix amb l'objectiu d'automatitzar l'avaluació d'estudiants sobre el temari pràctic de *Bases de Dades Relacionals*, sense la intervenció immediata de personal docent, mitjançant un procés d'auto-correcció. L'*Authoring Tool* en si és l'eina que permet al professorat configurar els exercicis, tant a nivell d'enunciat com a nivell de paràmetres necessaris pel procés d'auto-correcció, que més tard els alumnes resoldran. Donada l'àmplia varietat de tipus d'exercicis que es poden plantejar sobre bases de dades relacionals, la finalitat de tot el sistema *LearnSQL*, l'auto-correcció, fa que l'*Authoring Tool* sigui una eina complexa, tant en lògica interna com en usabilitat, però molt ben solucionada, contemplant i donant cabuda a totes les possibilitats, oferint les eines de configuració necessàries per cada exercici.

El sistema *LearnSQL* actualment té un efecte molt positiu en el dia a dia del professorat, ja que permet desvincular-lo d'un feixuc procés de correcció (exercici per exercici, alumne per alumne) i alhora l'ajuda a mantenir un repositori amb exercicis que es poden consultar i/o re-aprofitar al llarg dels anys. L'estudiant també en surt beneficiat al poder tenir *feedback* instantani de la seva solució a cada exercici. No obstant, amb l'ús, és habitual detectar problemes, limitacions i aspectes a millorar en el sistema. En el cas de l'*Authoring Tool* es fan evidents limitacions de disponibilitat i distribució de l'eina. Aquesta ha d'estar prèviament instal·lada a les màquines on es vol usar, que també han de tenir instal·lat *JAVA JRE*. A l'hora de distribuir una nova versió de l'eina, és difícil assegurar que arriba a totes les persones interessades, i aquestes s'encarreguen adequadament d'actualitzar-la. Aquest fet, potencialment, dóna lloc a tenir usuaris usant versions antigues (que inclouen errors crítics) una temporada, si no sempre.

En un altre context, la *plataforma web* en general destaca precisament, i per definició, en els camps de la disponibilitat i la distribució dels seus continguts. És per això, que des de la facultat es veu la necessitat de portar-hi l'*Authoring Tool*. Amb aquest nou paradigma, només és necessari un navegador web per accedir a (l'última versió de) l'eina. Així doncs, l'objectiu d'aquest projecte és portar l'*Authoring Tool,* amb totes les seves funcionalitats, cap la plataforma web.

En aquest projecte es prioritza la usabilitat i l'experiència d'usuari finals del producte. També es prioritza l'exploració d'eines àgils de desenvolupament que ens permetin aconseguir-ho. També es prioritza evitar que l'*Authoring Tool Web* esdevingui una solució tancada, limitada a desenvolupadors, i s'intenta que sigui propera al desenvolupament web quotidià, no acoblant la tecnologia usada al client (navegador) amb la tecnologia usada al servidor, permetent així la col·laboració d'altres rols professionals en el futur manteniment de l'aplicació (dissenyadors gràfics, front-end web developers, experts *UI/UX*[1], etc.), i permetent també que el nostre sistema pugui ser consumit per altres clients que no siguin el nostre.

Per al desenvolupament d'aquest projecte s'ha seguit un cicle de vida lineal amb les següents etapes. Primerament s'han assimilat els objectius del projecte. A continuació s'ha fet un estudi de l'eina existent (funcional, intern i d'usabilitat), del qual se'n beneficien les etapes posteriors. A aquesta alçada ja tenim tot el que necessitem per tenir clars els requeriments del projecte. Posteriorment s'ha fet recerca per tal de trobar la millor manera d'assolir els requeriments, tenint en compte les prioritats d'aquest projecte, donant pas a l'etapa de disseny. Amb el disseny del sistema clar, comença l'etapa d'implementació, en la que s'ha invertit bona part del temps prototipant fins a tenir un marc de treball prou bo, amb les funcionalitats transversals bàsiques, per poder continuar amb la implementació de les funcionalitats específiques de *l'Authoring Tool*. Un cop finalitzada la implementació, un usuari expert ha realitzat les proves d'acceptació corresponents per assegurar que els requeriments es compleixen, i, de no ser així, se n'han arreglat les parts afectades.

Gràcies a les premisses, i a unes intenses etapes d'estudi i recerca, el producte final obtingut preserva, i fins i tot pretén millorar, la bona experiència a la que els usuaris habituals de l'eina estan acostumats, sense sacrificar-ne funcionalitat. Això ha estat possible gràcies a l'adopció de les pràctiques més modernes en l'àmbit del desenvolupament d'aplicacions web. Un bon estudi previ també ens ha permès detectar i re-utilitzar bona part de la ja consolidada lògica de negoci de l'eina existent, reduint així

[1] *User Interfaces / User Experience*

al màxim la possibilitat de retrobar problemes coneguts ja resolts anteriorment. Les tecnologies que han tingut un paper clau en la construcció de l'*Authoring Tool We*b són *AngularJS*, *Flexbox*, *JSON*, i Apache *Struts2*.

Capítol 1: Introducció

1.1 Context del projecte

Aquest projecte parteix del projecte de final de carrera anterior *Gestor de cuestiones SQL y servicios web* correctores (Toporcer, 2007), posteriorment ampliat per un altre projecte anomenat *Desenvolupament de noves funcionalitats i millores de l'aplicació de gestió de qüestions de LEARN-SQL* (Fernàndez, 2010). En aquests projectes s'hi desenvolupa principalment un dels components del sistema *LearnSQL*: l'*Authoring Tool*, eina en la qual es centra aquest projecte.

1.1.1 *LearnSQL*

Recordem que *LearnSQL* és un sistema complex, format per diversos components, que beneficia tant a professors com a estudiants, permetent:

1. presentar exercicis pràctics en matèria de bases de dades relacionals a estudiants,
2. acceptar les solucions en forma de sentències *SQL* que els estudiants proposen,
3. corregir-les automàticament sense la intervenció directa de personal docent,
4. i proporcionar *feedback* immediat a l'estudiant.

1.1.1.1 Procés d'autocorrecció

Per poder validar automàticament una solució proposada per un estudiant, el sistema ha de tenir amb què comparar-la. Així doncs, és necessari poder especificar per cada exercici una solució model vàlida prèviament. No obstant, donada la flexibilitat del llenguatge *SQL*, és possible obtenir els mateixos resultats usant solucions diferents. És per això que el procés d'auto-correcció no pot dedicar-se només a comparar la solució proposada per l'alumne amb la solució model, sinó que requereix executar ambdues solucions en un entorn en iguals condicions i comparar-ne els resultats, donant prioritat als resultats de la solució model.

El procés de correcció normalment comença amb una base de dades buida, i es pot simplificar en les següents etapes:

1. Establir el context de l'exercici proporcionant a la base de dades una estructura o uns continguts inicials, necessaris per dur a terme l'exercici.
2. Executar la solució de l'exercici.
3. Comprovar que la solució proposada genera els resultats esperats.
4. Restaurar la base de dades.

No obstant, els exercicis plantejats poden ser de molts tipus diferents, com configurar l'estructura d'una base de dades, o canviar l'estat d'una base de dades modificant-ne les

dades que conté, o senzillament realitzar consultes per obtenir un subconjunt de dades determinades. Tots aquests casos diferents tenen diferents maneres de comprovar-se, i és per això que el sistema ha d'oferir una amplia varietat d'eines als professors per configurar els exercicis.

Quan es vol verificar que una solució és correcta, s'acostumen a plantejar diverses situacions que la posen a prova. Si la solució genera els resultats esperats en totes les situacions plantejades és signe que la resposta és bona. Amb aquesta finalitat existeixen els jocs de proves. No obstant, hi ha cops que les solucions dels exercicis no han de retornar cap resultat, com pot ser el cas d'un exercici de creació d'estructures en una base de dades. En aquest cas s'utilitzen les comprovacions de cada joc de proves, que permeten fer un anàlisis més de la situació.

Veiem doncs, que la tasca de configurar un exercici per a la seva posterior auto-correcció no és una tasca senzilla, i utilitza varies eines per cobrir totes les tipologies d'exercicis.

1.1.1.2 Components del sistema *LearnSQL*

Podem veure els components del sistema i les seves relacions a la *Figura 1.1*.

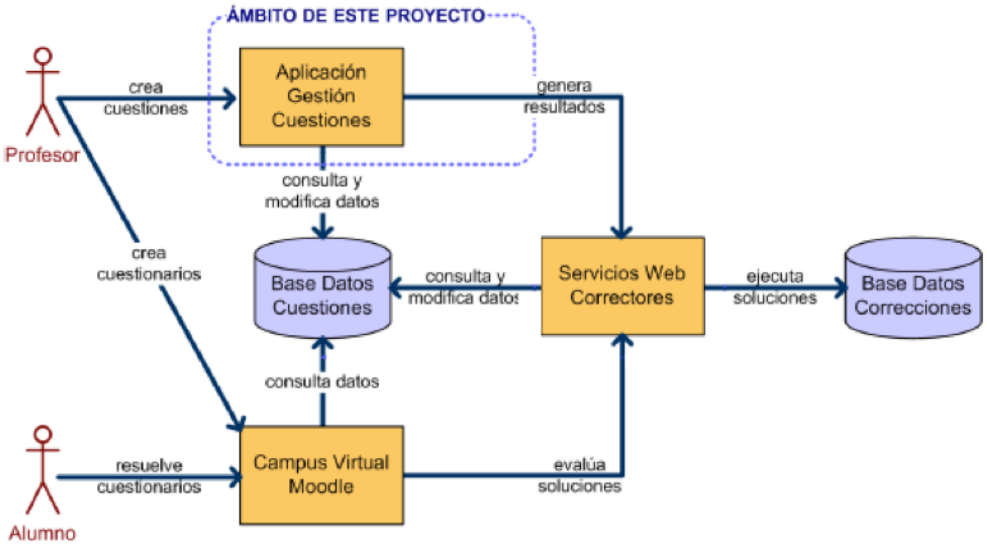


Figura 1.1 Components del sistema LearnSQL i les seves relacions

Aquest projecte es centra només en l'*Authoring Tool*, i en cap altre dels components del sistema *LearnSQL*.

Gestor de qüestions (*Authoring Tool*)

L'*Authoring Tool* és una aplicació d'escriptori construïda en *JAVA* que té la finalitat de donar accés al repositori central de qüestions del sistema i permetre gestiona-lo. Els seus usuaris són el personal docent de les assignatures de bases de dades. Els destinataris de les qüestions són els alumnes, que veuran les qüestions a través del mòdul de *Moodle* del sistema.

En més detall, l'eina està pensada per:

1. Configurar qüestions:
    a. Dades contextuals (títol, enunciat, etc.).
    b. Paràmetres necessaris per al procés d'auto-correcció (classificacions, sentències SQL, jocs de proves, comprovacions, etc.).
2. Executar i monitoritzar el procés de generació de resultats de la solució proposada pel professor de cada qüestió per a la futura auto-correcció de les solucions que proposarà l'estudiant. (*Veure 1.1.1.1*). L'aplicació usa els *serveis web correctors* per a aquest procés.

Donada l'àmplia varietat de tipus d'exercicis que es poden plantejar sobre bases de dades relacionals, la finalitat de tot el sistema *LearnSQL*, l'auto-correcció, fa que l'*Authoring Tool* sigui una eina teòricament complexa, tant en lògica interna com en usabilitat, però molt ben solucionada, contemplant i donant cabuda a totes les possibilitats, oferint les eines de configuració necessàries per cada exercici.

Mòdul de Moodle

Moodle és un programari de codi lliure que crea entorns virtuals d'ensenyament i aprenentatge.

L'extensió de *Moodle* creada específicament pel sistema *LearnSQL* permet usar la plataforma *Moodle* per a crear qüestionaris seleccionant diferents qüestions de les que hi ha emmagatzemades al repositori principal del sistema *LearnSQL*, prèviament creades amb l'*Authoring Tool*. Cada qüestionari pot tenir un o més exercicis de diferents categories i temàtiques. Per l'altra banda, permet als alumnes resoldre'ls.

Serveis web correctors

Els serveis web correctors són un sistema centralitzat que s'encarrega de la correcció dels exercicis amb la solució proposada.

Per una banda, l'*Authoring Tool* l'utilitza per generar les sortides de la solució proposada pel professor, que després s'utilitzaran per comparar-les amb les de l'alumne. En aquest mateix procés també es fa una validació de que totes les sentències *SQL* que han introduït els professors per configurar l'exercici són sintàcticament correctes, i es poden executar en l'ordre que indica la tipologia d'exercici.

Per l'altra, el mòdul *Moodle* utilitza els serveis per comprovar si la solució de l'alumne és bona.

1.1.2 Problemàtica

Amb l'ús continuat del sistema, i la necessitat de mantenir-lo, es fan evidents, entre d'altres, dos grans problemes de l'*Authoring Tool*:

1. **Disponibilitat**: només es pot accedir a les funcionalitats de l'aplicació en aquells dispositius on hagi estat prèviament instal·lada, i que puguin córrer aplicacions *JAVA*. Requereix anticipar-se.
2. **Distribució**: l'aplicació s'ha de donar a cada usuari que la vulgui usar, i, un cop se'n fa una nova versió, s'ha de tornar a distribuir. Aquest fet, potencialment, dóna lloc a tenir usuaris usant versions antigues (que contenen errors crítics o que no són compatibles amb el model de dades actual) una temporada, si no sempre.

És inevitable, doncs, donada la seva popularitat avui en dia, plantejar la migració cap a una altra plataforma que precisament destaca, per definició, en ambdós camps; la *plataforma web*.

D'altra banda, també hi ha altres aspectes que no són tan inherents a la plataforma actual, però que tampoc van tenir un resultat satisfactori en el desenvolupament inicial, com, per exemple, la pobra adaptació de la interfície gràfica a certs tipus de resolucions de pantalla.

1.2 Objectiu del projecte

Com hem dit prèviament, aquest projecte es centra només en el gestor de qüestions *Authoring Tool*.

Donada la problemàtica actual en disponibilitat i distribució de l'eina, i la creença que la plataforma web n'és la solució, el principal objectiu d'aquest projecte és, doncs:

Construir una aplicació web que:

- compleixi amb tots els requeriments dels projectes desenvolupats anteriorment,
- aprofiti el màxim possible de la lògica de negoci existent,
- millori alguns aspectes actualment no satisfactoris de l'aplicació existent.

Veure *Capítol 3* per a més detall.

1.3 Raó i oportunitat

En efecte, la plataforma web va ser dissenyada per oferir alta disponibilitat i fàcil distribució de continguts. Uns continguts, però, que, en un principi, havien de ser només documents estàtics indexables enllaçats, i no aplicacions complexes riques en interacció d'usuari. Aquest plantejament inicial és el causant de que encara avui en dia les aplicacions web estiguin molt lluny dels bons resultats que s'obtenen en aplicacions d'escriptori en quant a experiència final d'usuari i metodologies àgils de desenvolupament ben estructurades que ho facilitin.

Per sort, en els darrers anys, hi ha hagut una millora considerable en les funcionalitats estàndards del client web, i, sobretot, un *boom* per part de la comunitat en el desenvolupament de *frameworks* àgils que imposen metodologies i ofereixen funcionalitats vitals per desenvolupar aplicacions web de qualitat, sense allunyar-se del llenguatge de la web.

L'eina *Authoring Tool* és una aplicació d'escriptori complexa, però molt ben solucionada, amb una experiència d'usuari molt satisfactòria i còmoda. El nivell de complexitat que planteja, en quant a interfície gràfica (formularis, pestanyes, més formularis dinàmics niuats, indicadors que canvien a l'instant, flexibilitat, tirar enrere canvis, etc.) és notable, i, per desgracia, ja s'assumeix que aquesta experiència es veurà empobrida en la nova plataforma web.

A nivell personal, el meu interès per oferir una bona experiència d'usuari en general és cada cop més gran. Crec que aquest projecte és una bona oportunitat per explorar les

possibilitats d'aquestes noves tecnologies, cada cop més demanades en el món laboral, i provar-ne el seu abast, així com veure també com milloren el desenvolupament web convencional, sempre amb l'objectiu d'aconseguir un producte final amb una experiència d'usuari que com a mínim iguali a la del producte existent.

Així doncs, a banda de l'evident objectiu principal, aquest projecte també es marca el següent objectiu personal:

Construir una aplicació web que

- ofereixi una experiència d'usuari molt similar a la de l'aplicació d'escriptori existent, intentant reproduir-ne la majoria de "comoditats" que es donen per garantides en una aplicació d'escriptori, però que costa trobar en una aplicació web convencional,
- per fer-ho, explori noves tecnologies que ho afavoreixin, sense allunyar-se del llenguatge de la web.

1.4 Abast del projecte

En aquest projecte s'ha prioritzat la usabilitat i experiència d'usuari finals del producte. També s'ha prioritzat l'exploració de noves eines de desenvolupament que ens han permès aconseguir-ho.

Intentar reproduir i millorar l'experiència de l'eina de gestió existent sobre la plataforma web no és trivial, i ens condueix, per una banda, a construir un client a mida en la seva major part, i per l'altra, a adaptar la lògica de negoci de l'eina existent per a què respongui a les peticions del client. Per aquest motiu, la càrrega total de treball que suposava implementar totes les funcionalitats de l'*Authoring Tool* original era excessiva per a un únic projecte.

Així doncs, aquest projecte s'ha centrat en tenir un primera versió de l'*Authoring Tool Web* amb totes les funcionalitats de gestió de les principals entitats del sistema (qüestions, categories, temàtiques i esquemes), i serà responsabilitat de futurs projectes afegir-hi la interfície que dóna accés a la generació de resultats per al procés d'auto-correcció (gestió de correctors i execucions).

1.5 Tecnologies usades

En aquest projecte s'ha construït una aplicació web del tipus *SPA* (*Single Page Application*), cosa que suposa desenvolupar dues aplicacions: l'aplicació de la banda del client i l'aplicació de la banda del servidor.

Pel que fa a l'aplicació del servidor, s'ha utilitzat el *framework* d'aplicacions web *JAVA Apache Struts2* per estructurar-la i dotar a la lògica de negoci existent de la capa de serveis web necessària per atendre les peticions de l'aplicació client. La capa de persistència segueix sent un *SGBD PostgreSQL*.

Pel que fa a l'aplicació del client, s'ha utilitzat les tecnologies natives del navegador (HTML, CSS, Javascript) per construir-la, i el framework *AngularJS*, que ens ha ajudat a estructurar el codi font en components i a simplificar les tasques de refresc d'aquests components quan hi ha interacció d'usuari. Per altra banda, s'ha usat la nova especificació de *CSS*, *Flexbox*, per posicionar i distribuir els elements gràfics d'una manera molt més flexible i adaptable a diferents mides de pantalla.

La comunicació entre ambdues aplicacions es fa sobre el protocol *HTTP(s)*, usant el format d'intercanvi *JSON*.

Capítol 2. Estudi de l'aplicació existent

Aquest projecte parteix d'una aplicació ja existent i consolidada. Conèixer-la en profunditat abans d'avançar en el procés de desenvolupament de la nova versió és primordial per obtenir uns bons resultats finals. Amb aquesta mentalitat, s'ha fet un estudi previ exhaustiu de l'aplicació d'escriptori que busca:

- Entendre el seu context i la seva raó de ser.
- Entendre les funcionalitats que ofereix, i tenir-les en compte.
- Entendre el seu disseny i implementació, i tenir-lo en compte.
- Avaluar la seva usabilitat i la manera en la que els usuaris estan acostumats a interactuar-hi, i tenir-ho en compte.
- Detectar-hi problemes per poder plantejar-hi solucions.

Fixant-nos en aquests objectius, s'ha dividit l'estudi en quatre parts:

1. Estudi inicial
2. Estudi funcional
3. Estudi intern
4. Estudi d'usabilitat

Els objectes d'estudi han estat les memòries dels dos projectes anteriors a aquest, (Toporcer, 2007) i (Fernàndez, 2010), l'última versió del codi font (2.6), i l'aplicació en funcionament.

A continuació es detallen cadascun dels estudis esmentats, exceptuant l'estudi inicial, que tracta, senzillament, d'entendre el context i la raó de l'aplicació existent.

2.1 Estudi funcional

Aquest estudi analitza les funcionalitats que l'aplicació existent ofereix, amb la intenció d'usar-les més endavant en la definició clara dels objectius d'aquest projecte.

Les funcionalitats que s'observen es poden agrupar en buit paquets lògics principals:

1. Gestió de qüestions
   1.1. Gestió de jocs de proves
      1.1.1. Gestió de comprovacions
2. Gestió de categories
3. Gestió de temàtiques
4. Gestió d'esquemes
5. Gestió de correccions
6. Generació de resultats
7. Accés a l'aplicació
8. Altres

Es pot observar que tant els paquets lògics, com les funcionalitats detectades que contenen, coincideixen majoritàriament amb els casos d'ús especificats als projectes previs.

1. Gestió de qüestions

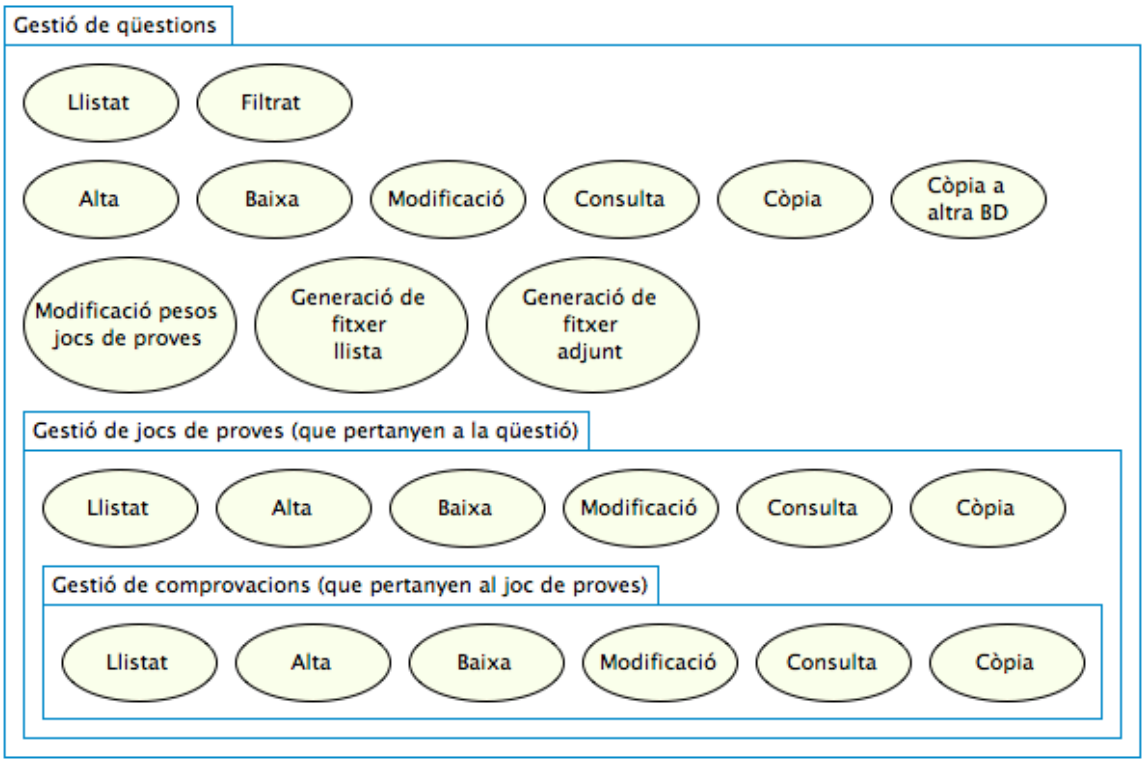


Figura 2.1 Funcionalitats del paquet "Gestió de qüestions"

2. Gestió de categories

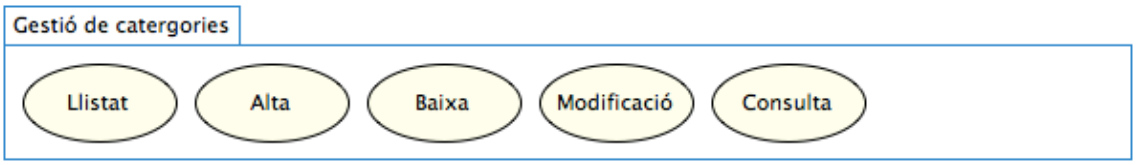


Figura 2.2 Funcionalitats del paquet "Gestió de categories"

3. Gestió de temàtiques

- Llistat
- Alta
- Baixa
- Modificació
- Consulta

4. Gestió d'esquemes

- Llistat
- Alta
- Baixa
- Modificació
- Consulta

5. Gestió de correctors

- Llistat
- Alta
- Baixa
- Modificació
- Consulta

6. Generació de resultats

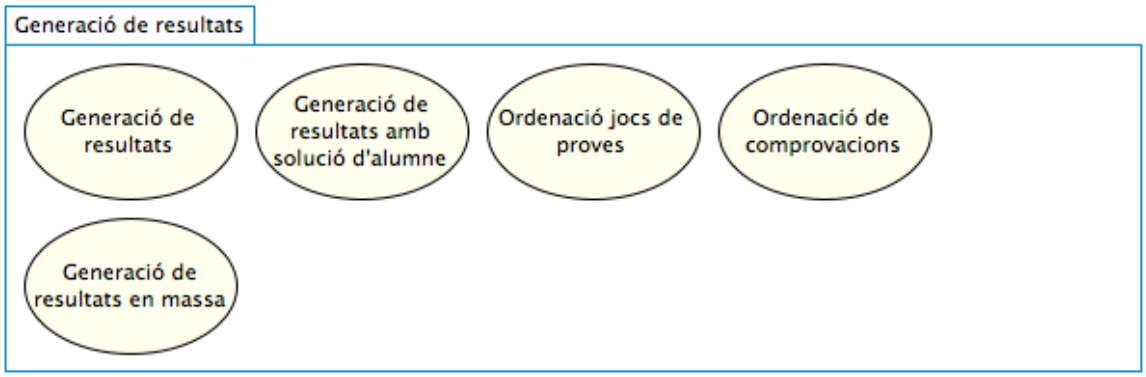


Figura 2.3 Funcionalitats del paquet "Generació de resultats"

7. Accés a l'aplicació

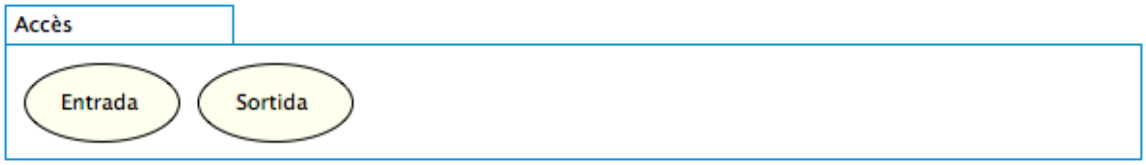


Figura 2.4 Funcionalitats del paquet "Accès"

8. Altres

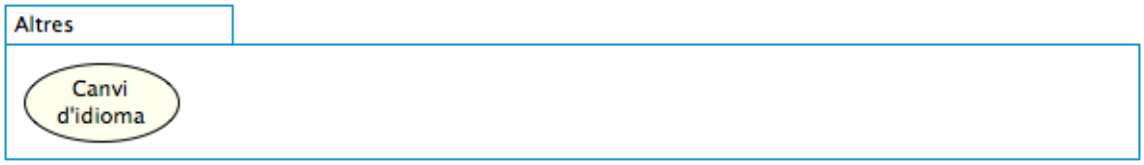


Figura 2.5 Funcionalitats del paquet "Altres"

2.2 Estudi intern

Aquest estudi documenta el disseny lògic i la implementació de l'aplicació existent. És important conèixer el projecte internament abans de poder decidir:

- de quina manera l'adaptarem a les nostres necessitats,
- quines parts n'aprofitarem, i quines no,
- quins nous desenvolupaments s'hauran de fer,
- quines tecnologies podrem usar.

Avís. Per veure com s'han alterat el disseny i la implementació per satisfer les necessitats d'aquest projecte, veure *Capítol 5. Disseny i implementació: Servidor*.

2.2.1 Arquitectura

L'aplicació existent està estructurada en la ja habitual arquitectura en 3 capes, que al afavorir el desacoblament de responsabilitats, contribueix a la bona comprensió i manteniment de l'aplicació.

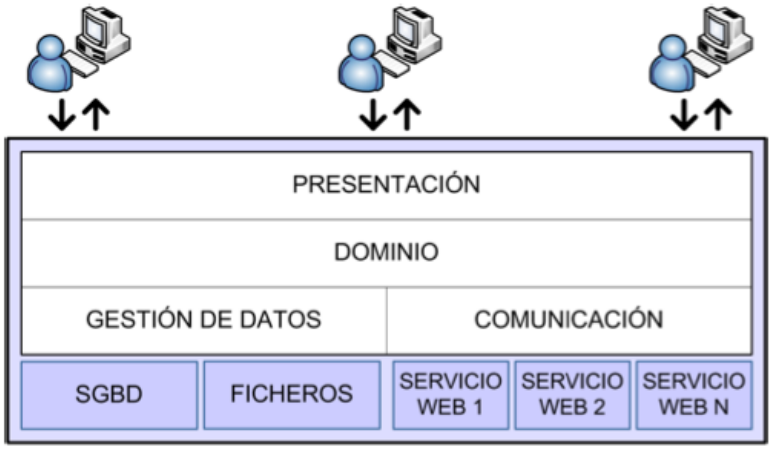


Figura 2.6 Arquitectura de l'aplicació existent

- **Capa de presentació**: es responsabilitza de la interacció amb l'usuari, tant per recollir les seves accions com per mostrar l'estat de l'aplicació.
- **Capa de domini**: conté la lògica de negoci de l'aplicació.
- **Capa de gestió de dades**: és l'encarregada de conèixer el mètode de persistència de l'aplicació, i fer-la possible.

En aquest cas hi ha una altra capa addicional, la capa de comunicació, que és l'encarregada de comunicar-se amb els serveis externs de correcció.

2.2.2 Model conceptual de dades

A continuació veiem com estan modelades les dades a gestionar per l'aplicació existent. Ens dona una idea de les entitats que conté el sistema i com es relacionen entre elles.

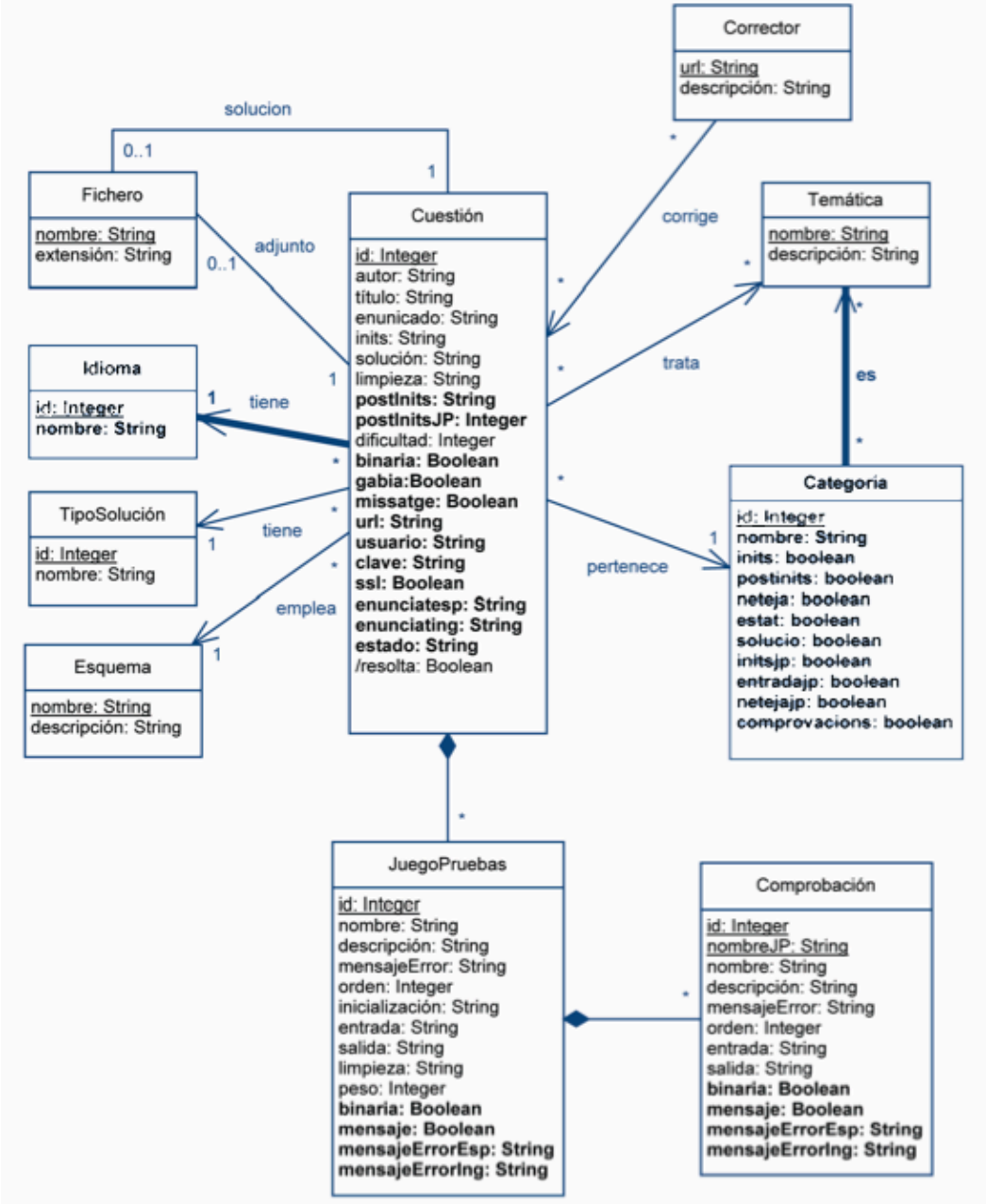


Figura 2.7 Model conceptual de dades de l'aplicació existent.

Restriccions textuals d'integritat:

- No hi pot haver dues qüestions amb el mateix identificador.
- No hi pot haver dos tipus de solució amb el mateix identificador.
- No hi pot haver dos idiomes amb el mateix identificador.
- No hi pot haver dues categories amb el mateix identificador.
- No hi pot haver dues temàtiques amb el mateix nom.
- No hi pot haver dos esquemes amb el mateix nom.
- No hi pot haver dos correctors amb la mateixa URL.
- No hi pot haver dos jocs de proves amb el mateix nom assignats a la mateixa qüestió.
- No hi pot haver dues comprovacions amb el mateix nom assignades al mateix joc de proves.
- Els ordres dels jocs de proves assignats a una qüestió han de mantenir valors correlatius.
- Els ordres de les comprovacions assignades a un joc de proves han de mantenir els valors correlatius.
- La dificultat de les qüestions només pot prendre valors entre 0 i 1.
- El pes dels jocs de proves només pot prendre valors entre 0 i 1.
- La suma dels pesos dels jocs de proves assignats a una qüestió ha de sumar 1.
- L'atribut derivat "/resolta" s'obté de la següent forma:
    - Una qüestió es considera resolta sempre que tingui jocs de proves assignats i cap d'aquests, ni tampoc cap de les seves possibles comprovacions, disposi d'una sortida buida.
    - En qualsevol altre cas la qüestió es considera no resolta.

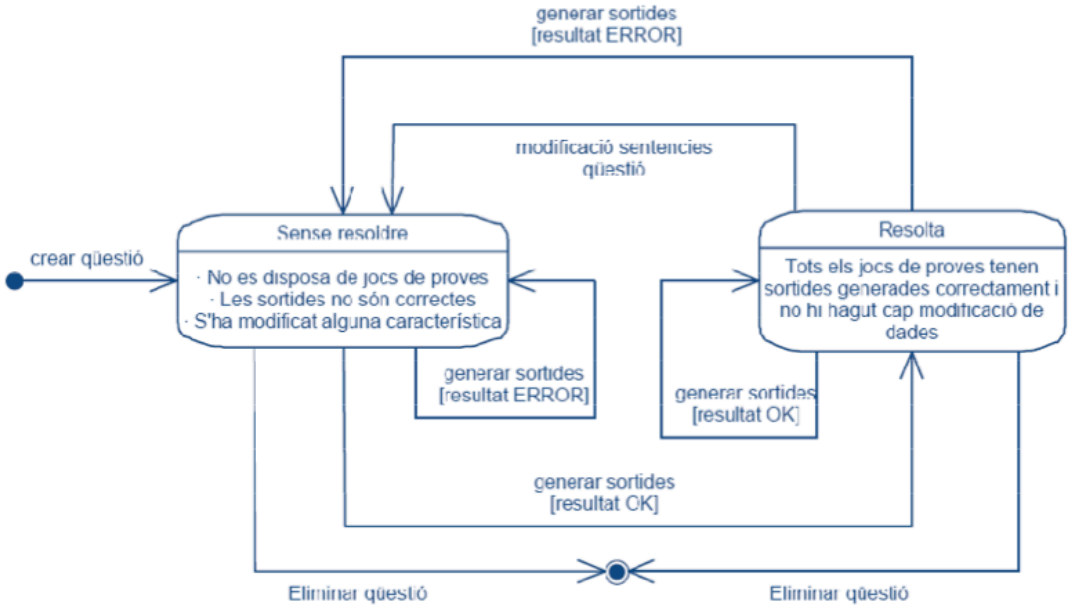


Figura 2.8 Model d'estats en els que pot estar una qüestió.

2.2.3 Implementació

2.2.3.1 Presentació

La implementació de la interfície d'usuari de l'aplicació existent s'organitza mitjançant la següent estructura jeràrquica de *Panels*:

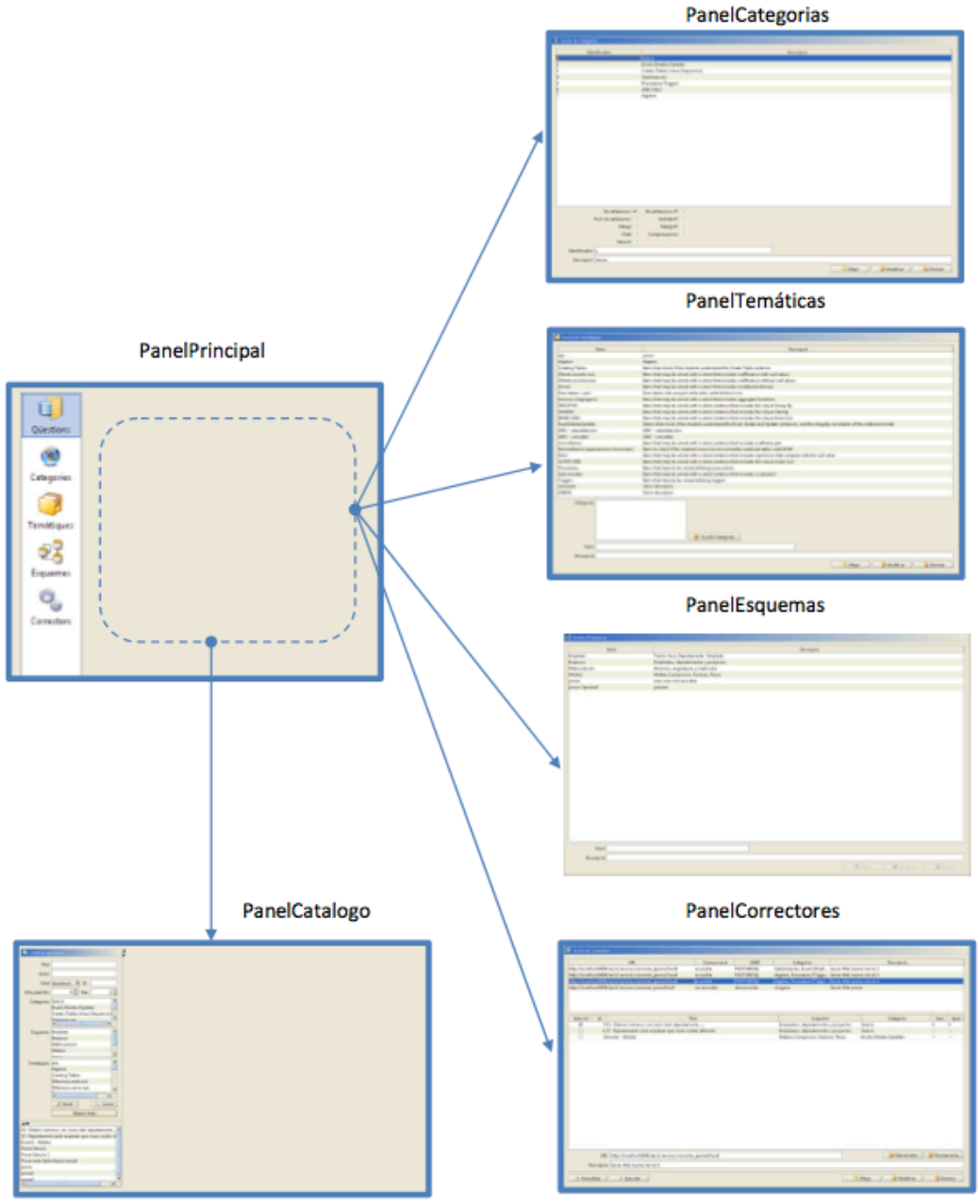


Figura 2.9

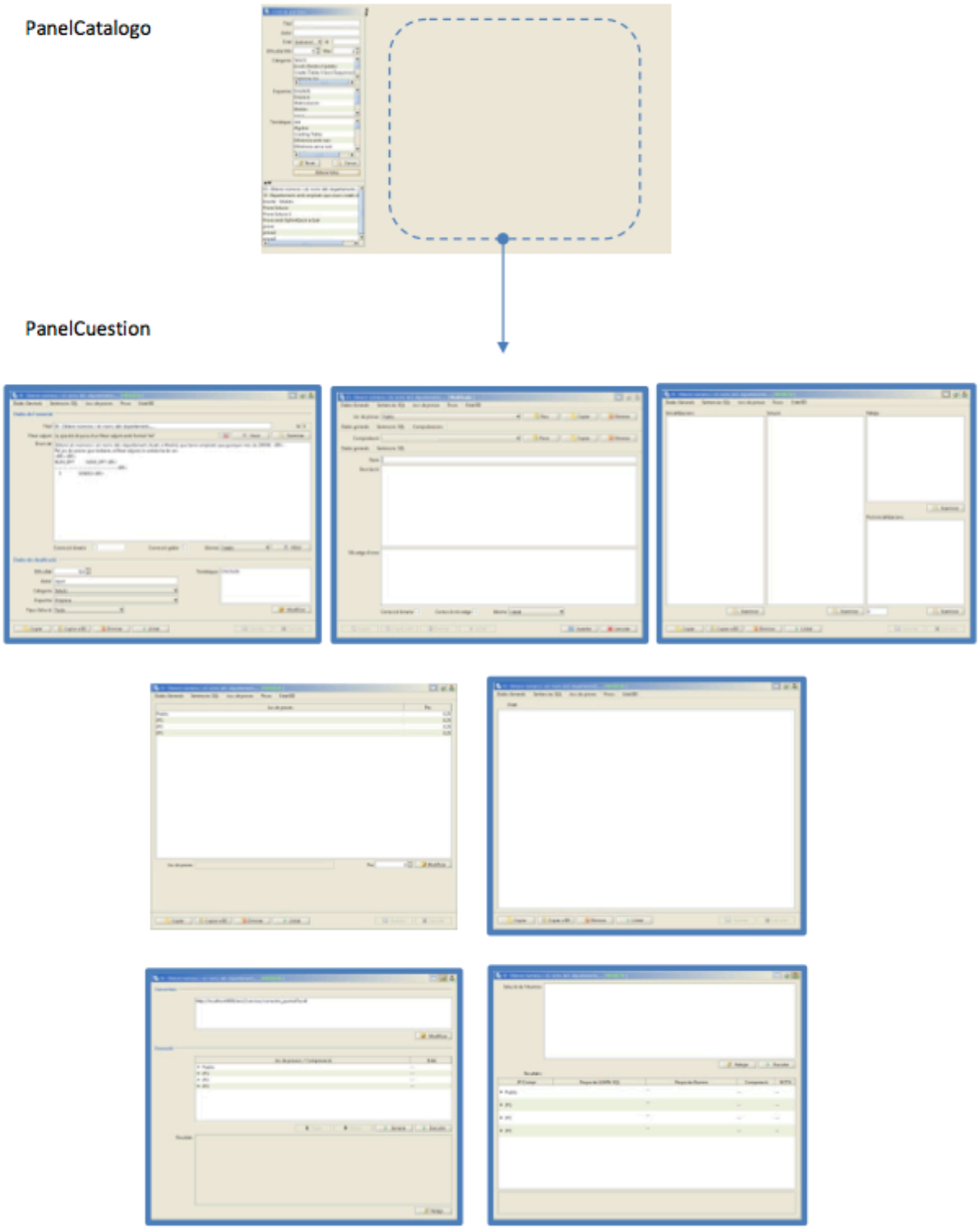


Figura 2.10

2.2.3.2 Domini

La capa de domini de l'aplicació existent la formen les entitats o models que corresponen al model conceptual de dades exposat anteriorment, i les classes diccionari corresponents a aquestes entitats, encarregades de, per una banda, mantenir les col·leccions de dades que l'aplicació fa servir (model de dades), i, per una altra, delegar operacions de consulta i persistència a la capa de gestió de dades quan el cas d'ús ho requereixi.

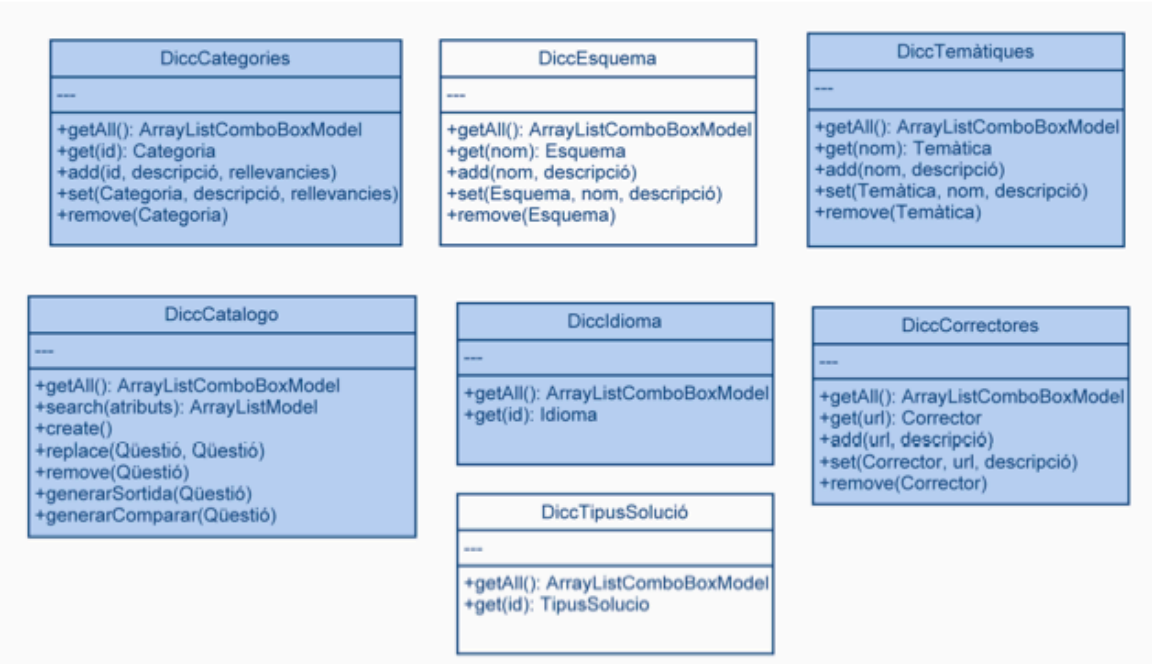


Figura 2.11 Diccionaris de la capa de domini de l'aplicació existent.

2.2.3.3 Gestió de dades

La capa de gestió de dades de l'aplicació existent està constituïda, majoritàriament, per les classes `Ctrls`, que s'encarreguen d'encapsular les sentències *SQL* que van a parar a la base de dades. Aquestes classes es connecten a la base de dades mitjançant l'objecte `Connection` de la llibreria *JDBC*.

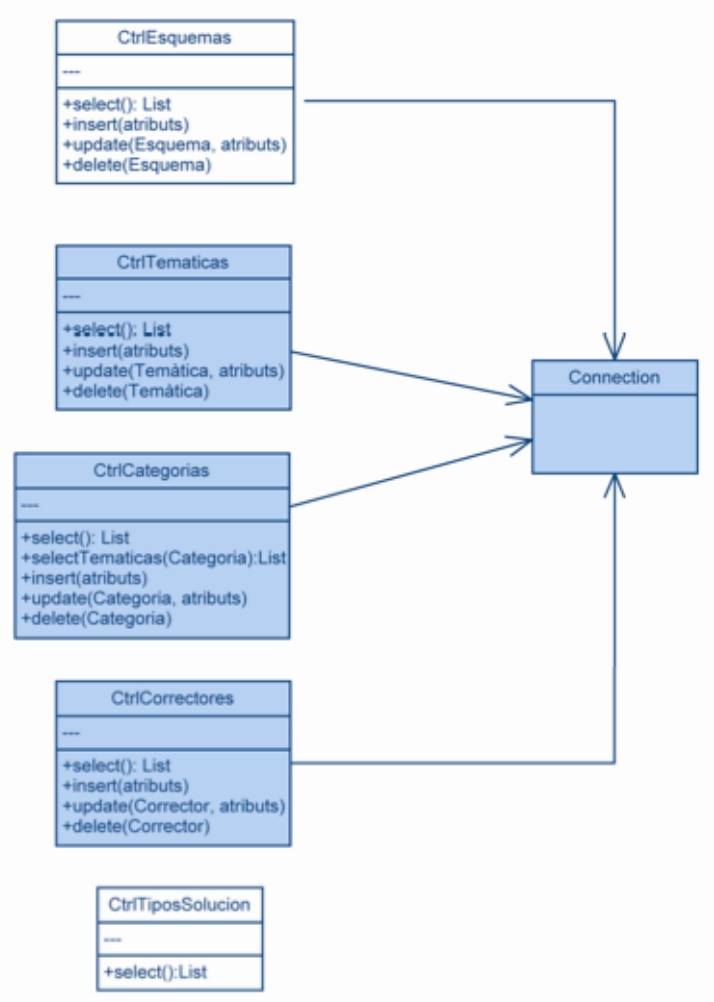


Figura 2.12 Classes Ctrl de l'aplicació existent i la seva dependència amb l'objecte Connection de JDBC

En el cas de les questions, a més a més, s'utilitza una classe ajudant anomenada `SaveOnCommit`. Aquesta classe facilita el maneig de la qüestió mentre s'està modificant, i persisteix o tira enrere els canvis quan se li demana, usant la capa de gestió de dades segons convingui.

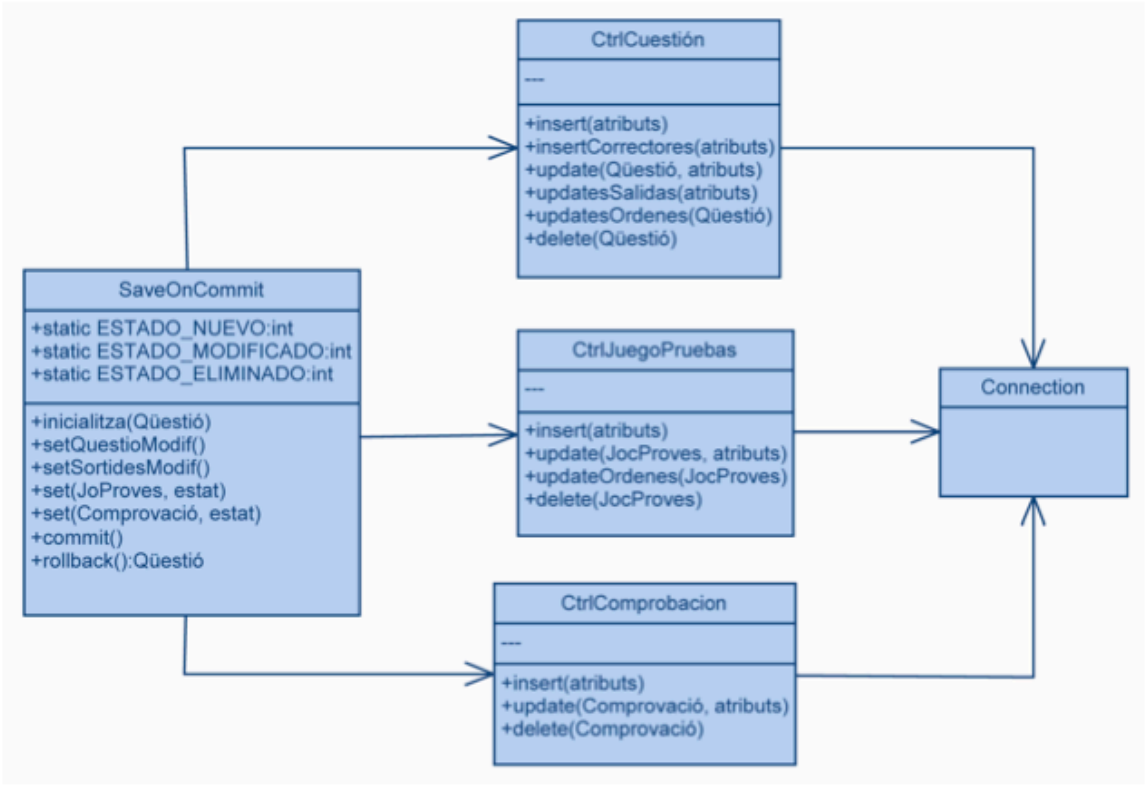


Figura 2.13 Classes Ctrl de l'aplicació existent relacionades amb una qüestió.

Finalment, aquest és el model conceptual de dades transformat en esquema de la base de dades.

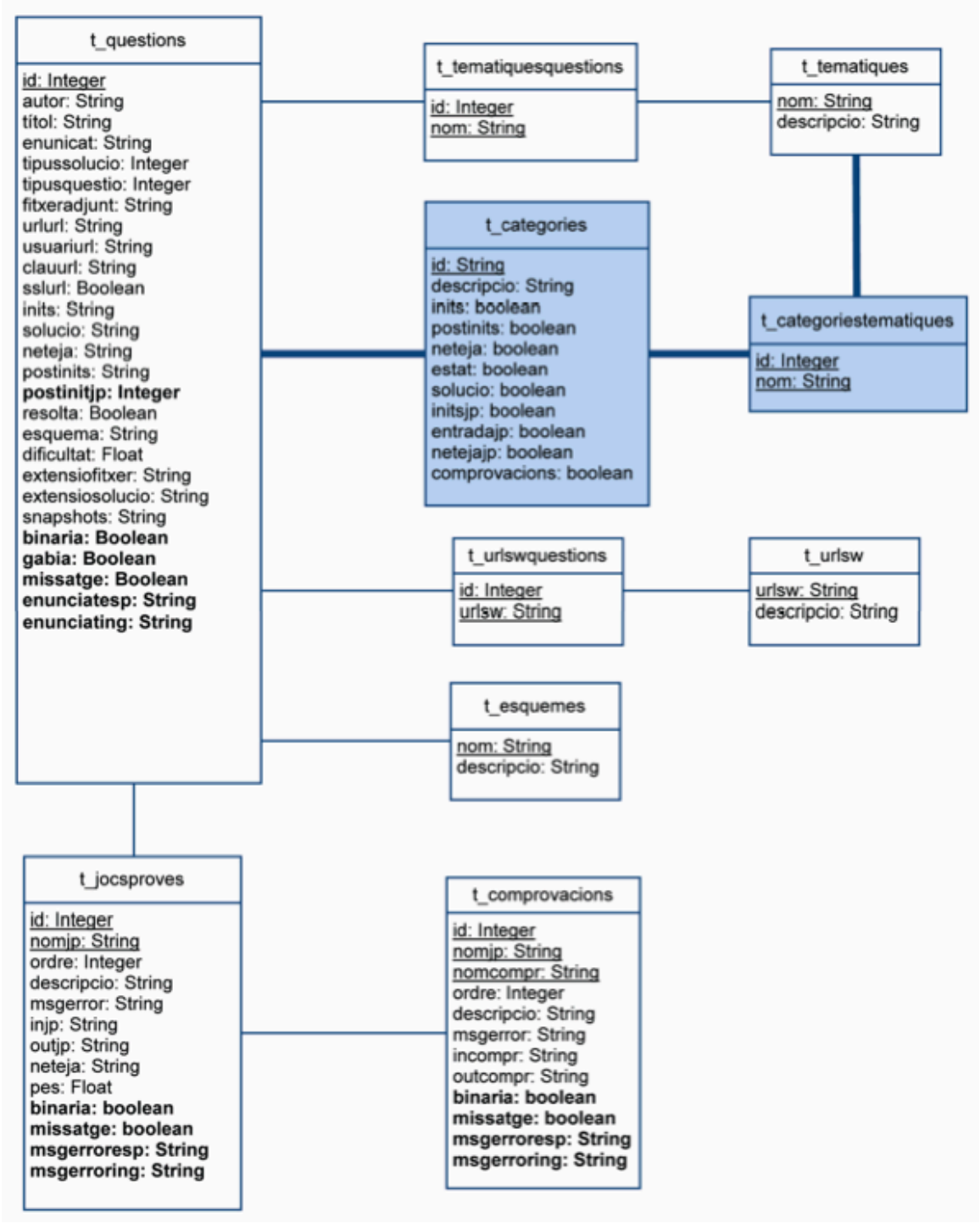


Figura 2.14 Esquema de base de dades de l'aplicació existent.

2.2.4 Tecnologies

Les tecnologies que s'usen en la implementació de l'aplicació d'escriptori existent són principalment *JAVA SE*, *JAVA Swing* per a la capa de presentació, i *JDBC* per la connexió amb l'*SGBD*, que actualment és *PostgreSQL*.

2.2.5 Conclusions de l'estudi

El disseny i implementació de l'aplicació existent solucionen correctament les necessitats de l'aplicació, i són producte d'un procés d'ús i refinat que asseguren la seva fiabilitat i correcte funcionament.. Aquestes mateixes necessitats encara existeixen en el nou projecte, per tant, no hi ha necessitat de re-inventar-les. El model conceptual de dades no canvia, i la lògica de negoci i la comunicació amb la base de dades es poden aprofitar en la seva major part.

2.3 Estudi d'usabilitat

Aquest estudi documenta la usabilitat de l'aplicació existent, anotant-ne les seves virtuts, i també els seus defectes, amb la finalitat de reproduir-ne les virtuts, i solucionar-ne els defectes, a la nova aplicació.

La bona percepció del producte final per part dels usuaris habituals de l'aplicació té molt a veure amb l'esforç invertit en aquesta etapa. És molt important entendre els seus hàbits per tal de no eliminar fluxos de treball als que estan acostumats.

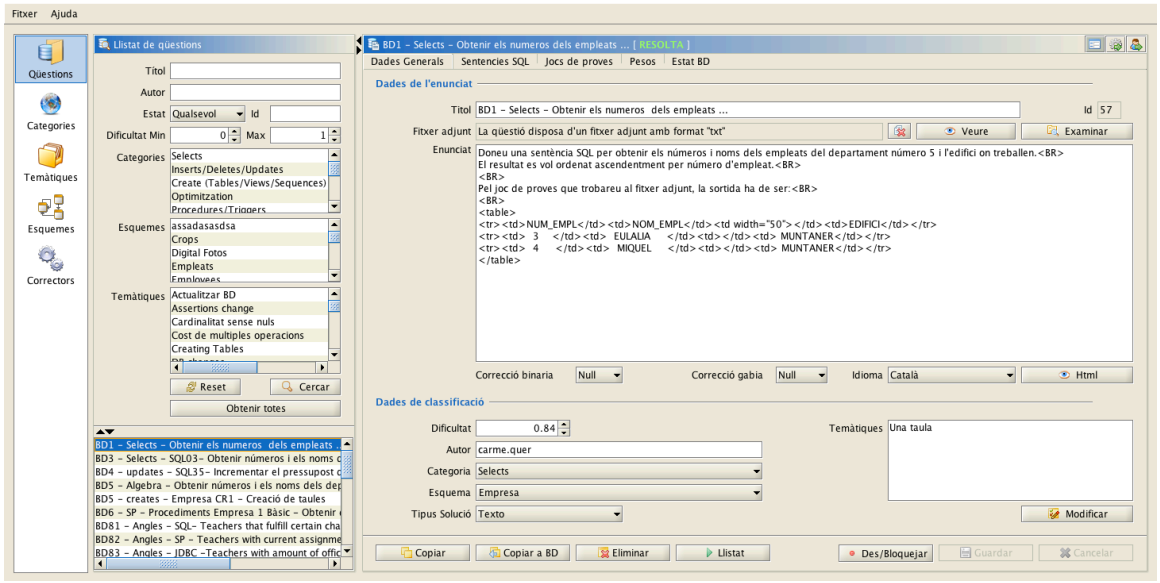


Figura 2.15 Distribució general de l'aplicació existent, per referència.

2.3.1 Virtuts

1 | Es manté l'estat, en general, del que s'està fent.

S'estigui fent el que s'estigui fent a l'aplicació, si marxem d'una secció o entitat en concret i hi tornem més endavant, ens ho trobem tot tal i com ho havíem deixat.

Aquesta és pot considerar la característica més important de totes, ja que habilita una infinitat de fluxos de treball, com per exemple:

- Marxar de l'entitat que estem consultant/creant/modificant, consultar-ne una altra per a comparar/copiar valors, i tornar finalment a l'entitat inicial per continuar amb el que s'estava fent.
- Marxar de l'entitat que estem consultant/creant/modificant, anar a una altra secció on es gestionen entitats relacionades, modificar-ne una, i tornar finalment a l'entitat inicial per continuar amb el que s'estava fent, tot veient com els canvis a l'entitat relacionada han afectat l'entitat inicial. Exemple: estem editant una qüestió que és de la categoria 1. Ens adonem que no se'ns deixa especificar el camp "inicialitzacions" per a la qüestió en concret, camp que hauria d'estar habilitat per a les qüestions d'aquesta categoria. Visitem la secció de categories, n'editem la categoria 1, i n'activem el camp "inicialitzacions". Finalment, tornem a la secció de gestió de qüestions, per trobar-nos la qüestió que estàvem editant, tal i com l'havíem deixat, però amb el camp "inicialitzacions" ara sí habilitat.
- Filtrar el llistat de qüestions per una sèrie de criteris, marxar de la secció qüestions, i tornar-hi finalment, veient que el filtratge es conserva.

2 | La interfície de navegació cap a les seccions principals sempre està a l'abast.

La manera en la que està distribuïda l'aplicació permet canviar de secció sempre que es desitgi, sense haver de fer passos addicionals, com per exemple, *scroll*.

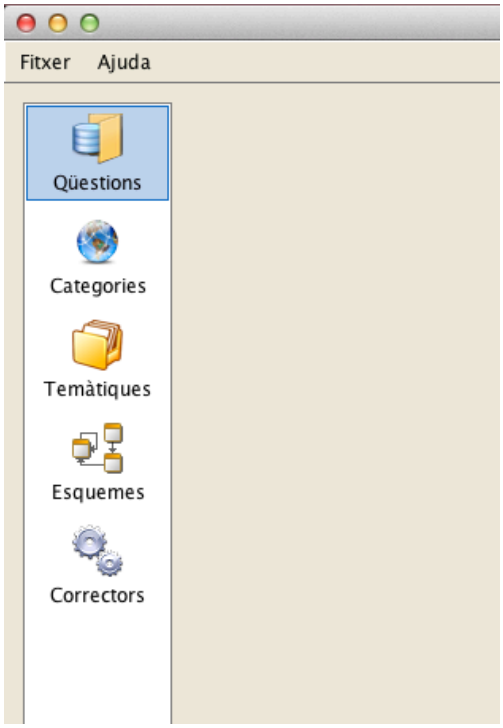


Figura 2.16 Menú de navegació a les seccions principals sempre a l'abast.

3 | Els llistats d'entitats sempre estan a l'abast mentre se'n consulta una, i és possible interactuar-hi.

Quan seleccionem una entitat des d'un llistat, amb la intenció de consultar-la/modificar-la, l'aplicació no fa desaparèixer el llistat, i per tant, un cop acabem la consulta/modificació, no se'ns obliga a interactuar amb el sistema per a tornar a veure el llistat altre cop. Al contrari, l'aplicació manté el llistat visible en tot moment, i és possible interactuar-hi amb la finalitat de consultar l'entitat que es vulgui, estiguem a l'entitat que estiguem, només en un sol pas.

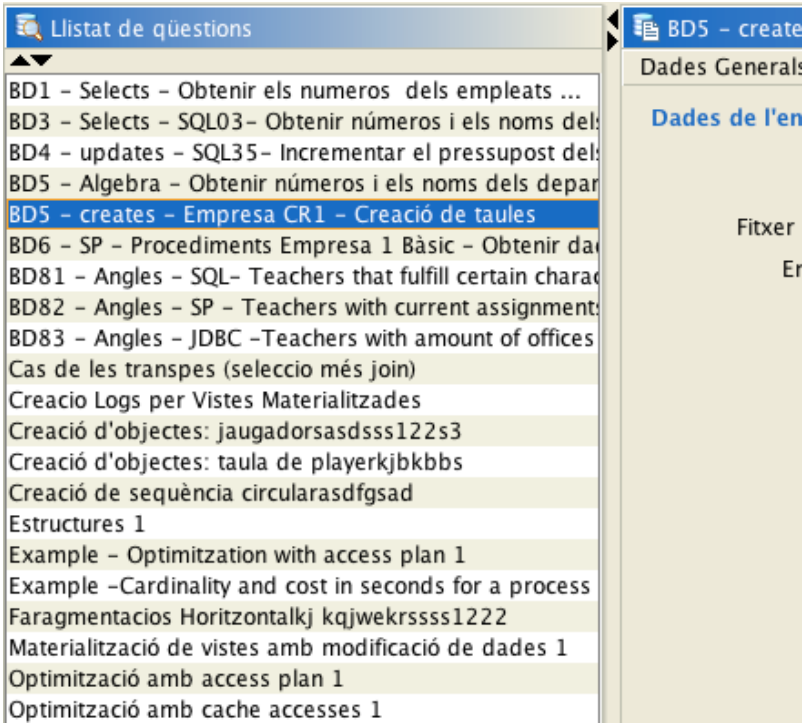


Figura 2.17

4 | Tots els elements importants relacionats amb el que s'està fent en un moment determinat estan visualment a l'abast.

Un bon exemple d'això són els botons d'accions que es poden executar sobre una entitat. Sempre estan situats a la part inferior de la finestra, independentment de la part de l'entitat que estiguem veient.

Un altre bon exemple és la manera en la que està organitzat el formulari d'una qüestió. Com que tots els camps no hi haguessin cabut en una sola pantalla, el formulari s'organitza en diferents seccions (pestanyes), assegurant així que quan estem veient una secció tenim en rang de visió tots els camps que aquesta ofereix.

La barra de títol d'una qüestió també n'és un bon exemple, ja que siguem on siguem de la qüestió ens permet recordar a simple vista quina qüestió estem veient, i també ens permet veure l'estat en el que està (Modificada/Resolta/No resolta/Bloquejada, etc.).

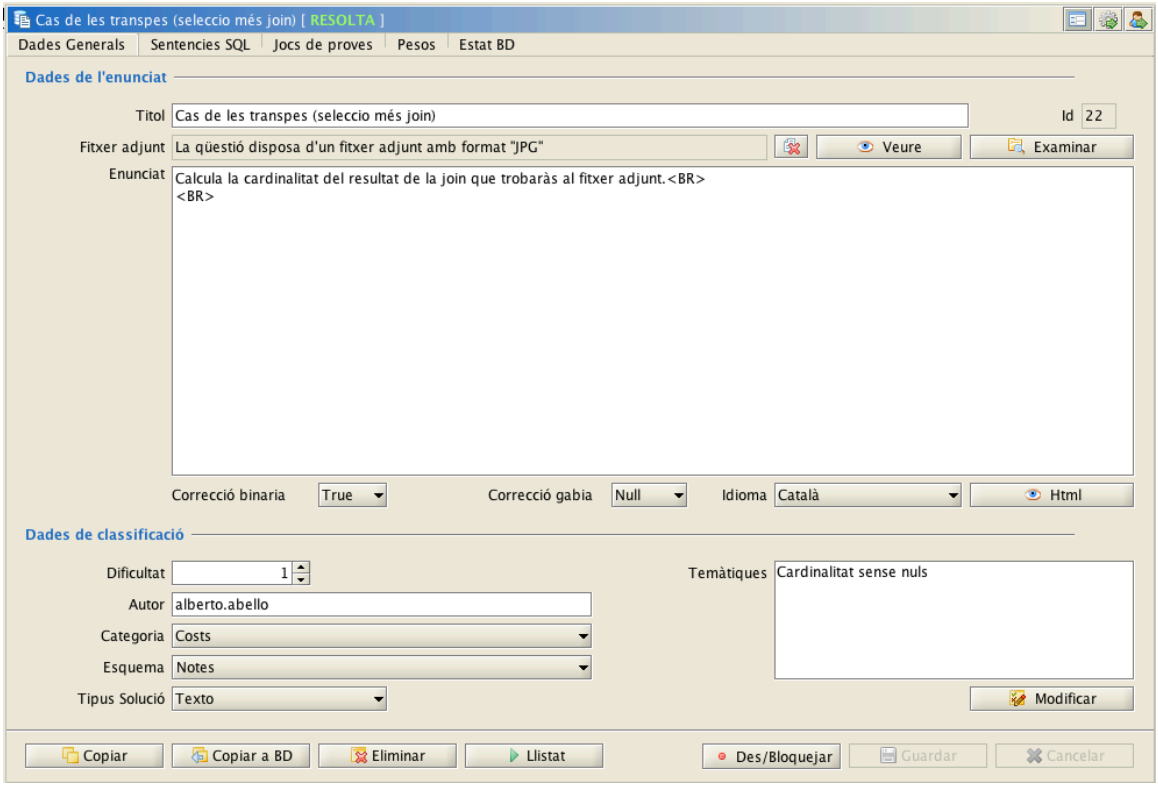


Figura 2.18 S'observen els botons d'accions inferiors, tots els camps de la secció, i la barra de títol, sempre visualment a l'abast.

5 | *Feedback* instantani. "Reactivitat".

Quan l'usuari fa accions transcendents, l'aplicació reacciona immediatament de manera visual per a què l'usuari entengui fàcilment la repercussió que aquestes accions tenen.

Un bon exemple és l'indicador d'estat d'una qüestió que hi ha a la seva barra de títol. Tan bon punt un usuari canvia el contingut d'algun dels camps de la qüestió, l'estat de la barra de títol canvia a "Modificada". Si tornem a deixar els camps de la qüestió tal i com estaven, l'estat de la barra de títol torna al seu estat anterior.

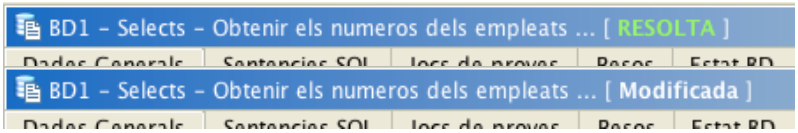


Figura 2.19

Un altre exemple són els botons d'accions d'una qüestió. Quan la qüestió no s'està modificant, es permet fer unes accions determinades, mentre que els botons de Guardar i Cancel·lar resten desactivats (no hi ha res a guardar o cancel·lar encara). Tan bon punt un usuari canvia el contingut d'algun dels seus camps, els botons de Guardar i Cancel·lar s'activen, i els de les altres accions es desactiven. En aquest moment, l'aplicació li està intentant dir a l'usuari que s'ha d'ocupar dels canvis que ha fet abans de fer cap altre cosa relacionada amb la qüestió. Un cop s'han desat o cancel·lat els canvis, els botons tornen al seu estat original.

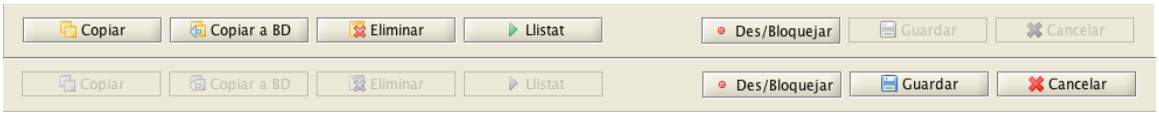


Figura 2.20

Un exemple més són els elements del formulari de qüestió que estan desactivats perquè la categoria actual no les permet. Quan l'usuari canvia la categoria actual a una altra categoria que sí els permet, aquests elements s'activen immediatament.

6 | Cancel·lació de canvis no confirmats.

L'aplicació manté la versió confirmada de cada entitat, i permet tirar enrere els canvis que s'hi hagin fet d'una manera immediata.

7 | Seguiment dels canvis efectuats sobre una qüestió.

L'aplicació anota quins canvis específics s'han efectuat sobre una qüestió, i, al voler-la confirmar o restaurar, l'aplicació informa a l'usuari de quins han estat els canvis, donant a l'usuari la seguretat suficient per fer el pas final.

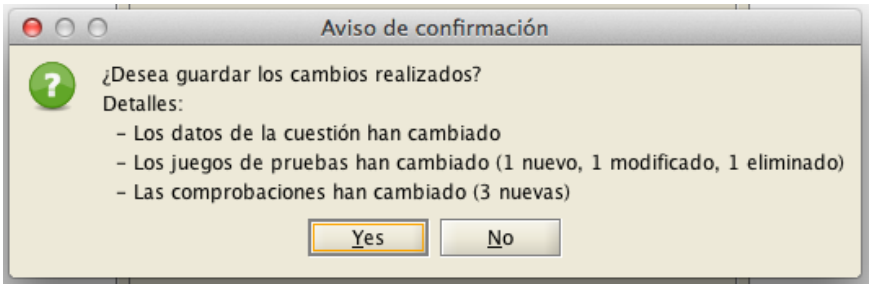


Figura 2.21

8 | Precisió en la comunicació d'errors de validació en una qüestió.

Quan hi ha un error de validació de dades al confirmar els canvis fets sobre una qüestió, l'aplicació presenta a l'usuari la secció exacta on es troba aquell error.

9 | S'aprofita tot l'espai visual disponible.

L'aplicació està pensada per ocupar sempre tot l'espai del que disposa. Tots els elements de l'aplicació estan distribuïts segons aquest criteri. Quan la finestra és més gran del compte, els elements que més espai necessiten s'expandeixen per aprofitar l'espai addicional.

Un bon exemple és l'àrea de text per especificar l'enunciat d'una qüestió. A mesura que l'espai vertical disponible augmenta, el camp enunciat creix verticalment per donar més visibilitat al seu contingut, mentre que els altres camps no canvien perquè no necessiten més espai.

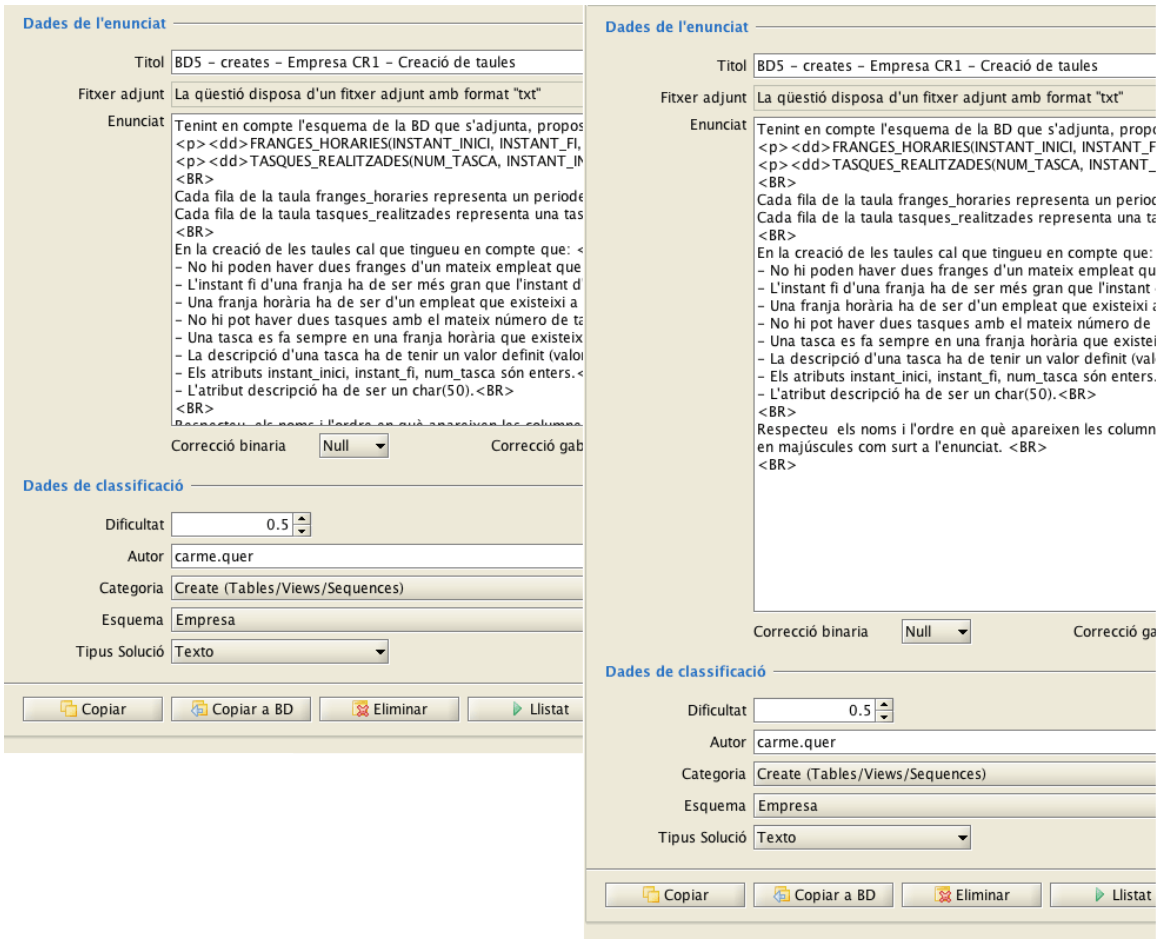


Figura 2.22

10 | Capacitat d'amagar elements per guanyar espai.

Un bon exemple és el component de filtrat de qüestions. Si el llistat de qüestions és llarg, tenim la possibilitat d'amagar el component de filtrat, i cedir l'espai que ocupava al llistat de qüestions.

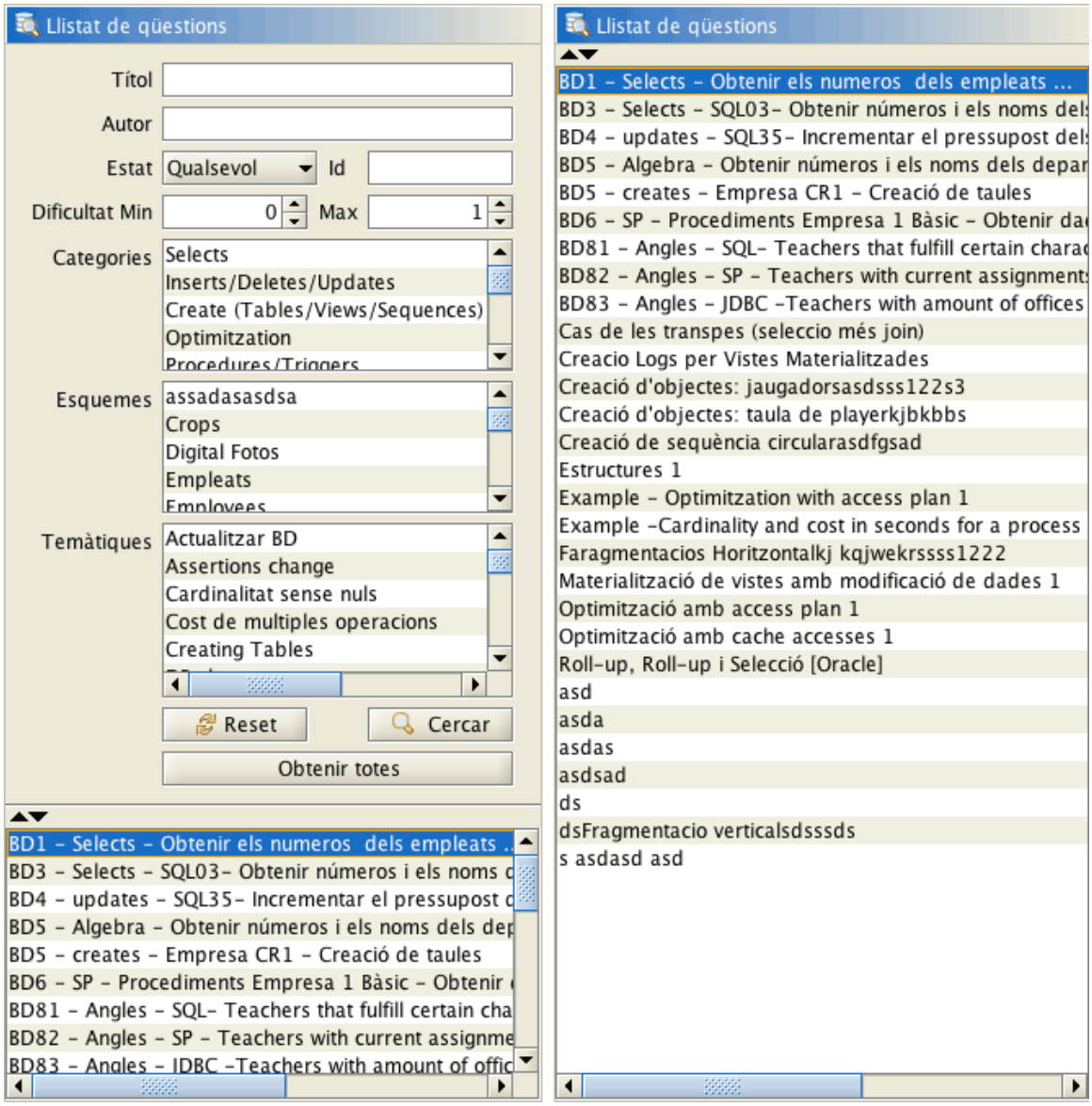


Figura 2.23

2.3.2 Defectes

1 | Alguns elements queden fora de la pantalla a resolucions inferiors a 1500x750.

L'aplicació va ser pensada per tenir tots els elements necessaris visualment a l'abast per una certa resolució de pantalla (o dimensió de finestra), però quan treballem amb dimensions més baixes, els elements comencen a no caber-hi, queden amagats fora de la finestra, i l'aplicació no proporciona cap mecanisme per accedir a ells.

Es pot dir que l'aplicació existent contempla la flexibilitat en expansió, però no en contracció.

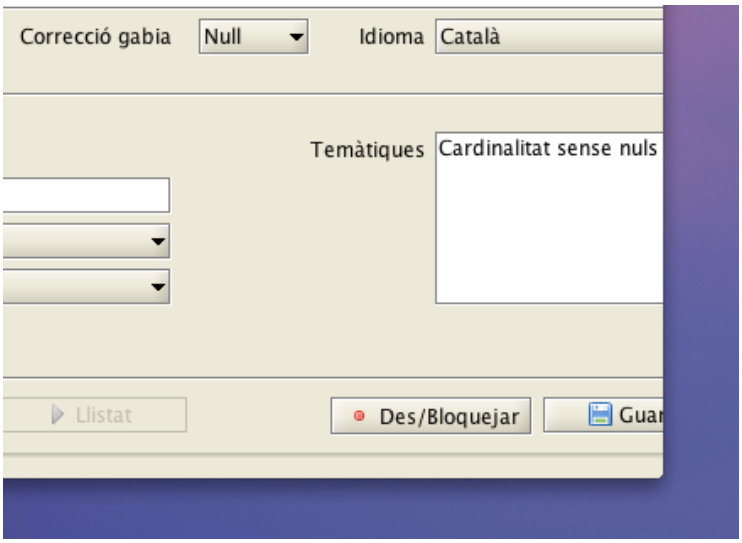


Figura 2.24

2 | No es veuen a simple vista tots els jocs de proves que té una qüestió, ni totes les comprovacions que té un joc de proves, quan s'estan consultant.

Quan estem gestionant els jocs de proves o les comprovacions d'una qüestió, no hi ha manera de veure a simple vista un llistat amb totes les que existeixen. L'única manera és desplegant el selector per triar-ne un, cosa que suposa una interacció addicional de l'usuari innecessària.

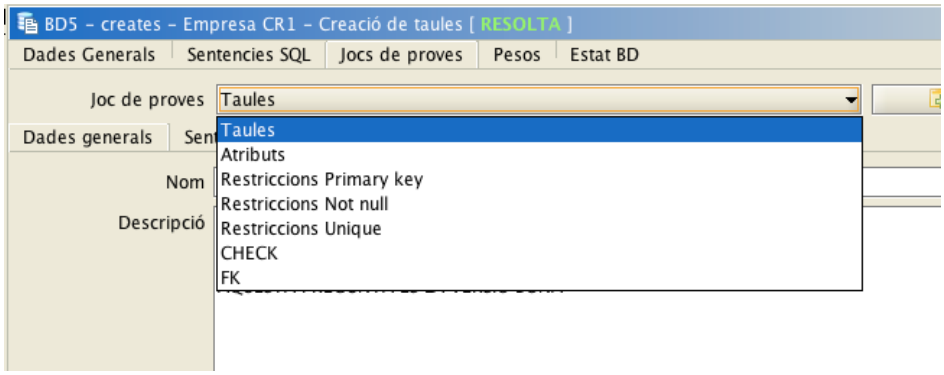


Figura 2.25

3 | No es veu a simple vista el sumatori del pes dels jocs de proves.

Tot i que que els pesos de tots els jocs de proves d'una qüestió sumin 1 és una condició indispensable per a poder persistir la qüestió correctament, i tot i que hi ha una secció únicament dedicada a mostrar els pesos de cada joc de proves, enlloc s'indica la suma total actual per a què l'usuari pugui saber si ha introduït els pesos bé o no.

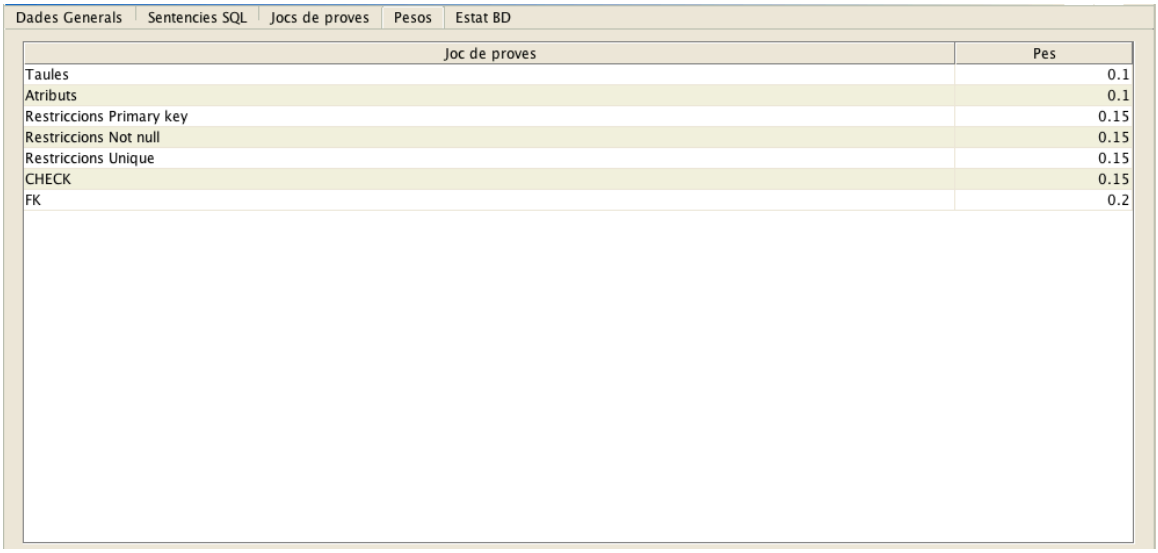


Figura 2.26

2.3.3 Conclusions de l'estudi

En general, la usabilitat de l'aplicació existent és molt bona, i soluciona molt bé la complexitat del sistema, permetent uns fluxos de treball molt còmodes, que no es volen perdre al renovar-la. Gràcies a aquest estudi, s'ha pogut percebre els aspectes essencials que no poden faltar a la nova aplicació web per tal de proporcionar una molt bona experiència d'usuari, i s'ha pogut veure quins aspectes cal solucionar.

Capítol 3. Anàlisi de requeriments

En aquest capítol es descriuen els requeriments del projecte, és a dir, les qualitats que el client espera del producte final.

En l'etapa inicial del projecte, prèvia a l'estudi de l'aplicació existent, ens reunim amb el nostre client, en aquest cas un client expert al tractar-se del professorat de bases de dades de la facultat, per analitzar les seves necessitats en vers al projecte. Al ser aquest un projecte majoritàriament de tipus *portabilitat*, la quantitat de funcionalitats noves que es necessiten és pràcticament inexistent, el que fa que la tasca d'anàlisi durant la reunió sigui breu, i no se'n pugui extreure immediatament un llistat de requeriments sense haver fet també prèviament un estudi funcional de l'aplicació existent[2], ni tenir coneixements de la futura plataforma. Això, però, no canvia el fet que el client hagi comunicat les seves necessitats amb precisió i claredat, i s'hagin entès perfectament. Aquestes necessitats, o, més aviat, aquesta necessitat és:

Necessitat del client

Construir una aplicació web que

1. ofereixi totes les funcionalitats de negoci dels projectes desenvolupats anteriorment;
2. aprofiti el màxim possible de la lògica de negoci existent;
3. utilitzi el mateix model de dades;
4. funcioni amb *PostgreSQL* versió 9.x (millora), a banda de la 8.x;
5. permeti escollir el repositori d'exercicis que es vol gestionar; (millora),
6. sigui totalment funcional en pantalles petites (millora);
7. sigui segura, eficient, mantenible, compatible, usable i fiable.

Amb aquests breus requeriments per part del client, un cop fet també l'estudi inicial funcional[2], i tenint coneixements de la futura plataforma, ja es pot elaborar l'anàlisi de requeriments més detallat, que trobem a continuació. Aquest anàlisis pot ser usat pel client com a referència a l'hora de fer les proves d'acceptació un cop entregat el producte.

[2] Veure *3.1 Estudi funcional*.

1. No funcionals

1.1 | L'aplicació s'ha de desplegar correctament en el servidor, mitjançant un contenidor web habilitat per a la tecnologia escollida.

1.2 | L'aplicació ha de funcionar correctament amb *PostgreSQL*, concretament amb les versions 8.x i 9.x

1.3 | L'aplicació ha de ser compatible amb els repositoris de qüestions existents.

1.4 | L'aplicació s'ha de poder connectar a les bases de dades que contenen els repositoris de qüestions, siguin a la mateixa màquina o no, i ho ha de poder fer usant la capa de transport segur *SSL*.

1.5 | Les dades de connexió a la base de dades han d'estar en un fitxer de text pla que pot ser editat fàcilment per un administrador de sistemes un cop desplegada l'aplicació. Aquestes dades són:

- SGBD
- Servidor
- Host
- Port
- SSL

1.6 | L'aplicació s'ha de poder connectar satisfactòriament als serveis correctors existents, siguin a la mateixa màquina o no.

1.7 | La comunicació Client-Servidor ha de poder usar el protocol HTTPS, per a la transmissió segura de dades.

1.8 | L'aplicació ha de demanar confirmació a l'usuari abans d'executar operacions importants.

1.9 | L'aplicació ha de validar l'input de l'usuari, assegurant-se que no s'introdueixen dades inconsistents al sistema.

1.10 | L'aplicació ha d'executar les operacions importants atòmicament, de tal manera que si hi ha un error, es conserva la integritat de dades anterior a l'operació.

1.11 | L'aplicació ha de mostrar un error si, per qualsevol motiu, no pot accedir al recurs/contingut que es demana.

1.12 | L'aplicació ha de ser usable en la majoria de dimensions de pantalla possible.

1.13 | L'experiència d'usuari és clara, intuïtiva i usable. Els usuaris de l'aplicació existent han de ser capaços de saber utilitzar la nova sense masses indicacions.

1.14 | L'aplicació ha de ser fàcilment mantenible, estant ben documentada, usant patrons coneguts tant en disseny com implementació, usant eines ben documentades, i deixant un codi font comprensible i ben estructurat.

2. Accés

2.1 | S'ha d'accedir a l'aplicació mitjançant una URL base.

2.2 | No es pot accedir a cap recurs de l'aplicació, exceptuant la pantalla d'accés, sense estar correctament autenticat.

- Intentar-ho acaba presentant la pantalla d'accés.

2.3 | Si el visitant no està autenticat, se li presenta una pantalla d'accés amb un formulari d'autenticació que conté els camps:

- Nom d'usuari
- Contrasenya
- Base de dades

Base de dades identifica quin repositori (base de dades/esquema) es gestionarà en la sessió actual.

2.4 | Els procés d'autenticació s'ha de fer contra els usuaris mateixos de la base de dades especificada (i no contra una taula d'usuaris).

- Si el procés falla, es notifica al visitant i es torna a presentar la pantalla d'accés altre cop.

2.5 | Un cop autenticat, l'usuari ha de poder finalitzar la sessió (sortir de l'aplicació/desconnectar) per voluntat pròpia clicant un botó.

2.6 | Quan la sessió s'acaba sense que l'usuari ho demani o es perd per qualsevol motiu, l'aplicació ha de notificar l'usuari i presentar la pantalla d'accés altre cop.

3. General

3.1 | L'aplicació s'ha de poder veure en Català (per defecte) i en Anglès, com a mínim. L'usuari ha de poder canviar l'idioma en tot moment.

4. Gestió de qüestions

4.1. Llistat de qüestions

4.1.1 | Ha d'existir un llistat de totes les qüestions.

4.1.2 | S'ha de poder pot filtrar el llistat de qüestions pels següents camps, sent possible combinar-los:

- Identificador
- Títol (parcial)
- Autor (parcial)
- Estat
- Categories
- Dificultat (min. i màx.)
- Temàtiques
- Esquemes

4.1.3 | S'ha de poder seleccionar una qüestió del llistat, per consultar-ne les seves dades.

4.2. Consulta de qüestions

4.2.1 Consulta de qüestions – General

4.2.1.1 | S'ha de veure el camp *identificador* de la qüestió.

4.2.1.2 | Ha d'existir un indicador que indica l'estat de la qüestió (Resolta / No resolta / Modificada / Bloquejada)

4.2.1.3 | S'han de mostrar els següents camps:

Dades generals

- Títol (text)
- Autor (text)
- Enunciat (textarea)
- Fitxer adjunt (file)
- Correcció binària (select)
- Correcció gàbia (select)
- Dificultat (number, 2 decimals)
- Categoria (select)
- Temàtiques (select múltiple)
- Esquema (select)
- Tipus solució (select)

Sentències SQL

- Inicialitzacions (textarea)
- Solució (mixed)
- Neteja (textarea)
- Post-inicialitzacions (textarea)
- Post-inicialitzacions JP (number)

Estat BD

- Estat BD (textarea)

4.2.1.4 | Camp *Enunciat*:

- Ha de permetre introduir contingut en 3 idiomes diferents (Català, Castella i Anglès).
- Ha d'existir una manera de previsualizar el seu contingut interpretat en HTML.

4.2.1.5 | Els camps *Correcció binària* i *Correcció gàbia* poden ser *NULL* (per defecte), *True,* o *False*.

4.2.1.6 | El valor del camp *Dificultat* ha d'estar entre 0 i 1, amb una precisió de 2 decimals. Per defecte és -1.

4.2.1.7 | Camp *Fitxer adjunt*:

- Si ja hi ha un fitxer:
  - s'ha d'indicar de quin tipus és (extensió).
  - s'ha de permetre obrir-lo.
  - s'ha de permetre permet substituir-lo, pujant-ne un altre.
  - s'ha de permetre eliminar-lo.
- Si no n'hi ha, s'ha de permetre pujar-ne un.

4.2.1.8 | Camp *Categoria:*

- Ha de permetre escollir una categoria d'entre d'entre totes les categories del sistema.
- Quan la categoria escollida canvia, s'ha de desactivar tots aquells camps de la qüestió (incloent jocs de proves i comprovacions) que no estan permesos per la categoria.

4.2.1.9 | El camp *Temàtiques* ha de permetre escollir N temàtiques d'entre totes les temàtiques que siguin de la categoria escollida al camp *Categoria*.

4.2.1.10 | El camp *Esquemes* ha de permetre escollir un esquema d'entre tots els esquemes del sistema.

4.2.1.11 | El camp *Tipus solució* ha de permetre escollir entre les opcions: *Text* (per defecte), *Fitxer adjunt*, o *URL+usuari+clau*. (Veure **4.2.1.14** per implicacions.)

4.2.1.12 | El camp *Inicialitzacions* ha d'estar desactivat si la categoria de la qüestió no permet aquesta característica.

4.2.1.13 | El camp *Solució* ha d'estar desactivat si la categoria de la qüestió no permet aquesta característica.

4.2.1.14 | El tipus del camp Solució varia en funció del valor del camp *Tipus Solució*:

- Si el camp *Tipus Solució* té el valor *Text*, el camp *Solució* només ha de permetre introduir-hi text.
- Si el camp *Tipus Solució* té el valor *Fitxer,* el camp *Solució* ha de ser tipus fitxer:
  - Si ja hi ha un fitxer:
    - s'ha d'indicar de quin tipus és (extensió).
    - s'ha de permetre obrir-lo.
    - s'ha de permetre permet substituir-lo, pujant-ne un altre.
    - s'ha de permetre eliminar-lo.
  - Si no n'hi ha, s'ha de permetre pujar-ne un.
- Si el camp *Tipus Solució* té el valor *URL+usuari+clau*, el camp *Solució* ha de mostrar els següents camps addicionals:
  - *URL (text)*
  - *Usuari (text)*
  - *Clau (text)*
  - *SSL (checkbox)*

4.2.1.14 | El camp *Neteja* ha d'estar desactivat si la categoria de la qüestió no permet aquesta característica.

4.2.1.15 | El camp Post-*inicialitzacions* ha d'estar desactivat si la categoria de la qüestió no permet aquesta característica.

4.2.1.17 | El camp *Estat BD* ha d'estar desactivat si la categoria de la qüestió no permet aquesta característica.

4.2.1.18 | S'ha de poder accedir als jocs de proves de la qüestió

4.2.2 Consulta de qüestions – Consulta de jocs de proves

4.2.2.1 | Ha d'existir un llistat de tots els jocs de proves de la qüestió.

4.2.2.2 | S'ha de poder seleccionar un joc de proves del llistat per consultar-ne les seves dades.

4.2.2.3 | Les dades que s'han de mostrar al consultar un joc de proves són:

Dades generals

- Nom (text)
- Descripció (textarea)
- Missatge d'error (textarea)
- Correcció binària (select)
- Correcció missatge (select)
- Consumeix intent (select)

Sentències SQL

- Inicialitzacions (textarea)
- Entrada (textarea)
- Neteja (textarea)
- Sortida (textarea)

4.2.1.4 | El camp *Missatge Error* ha de permetre introduïr contingut en 3 idiomes diferents (Català, Castella i Anglès).

4.2.1.5 | Els camps *Correcció binària*, *Correcció gàbia* i *Consumeix intent* poden ser *NULL* (per defecte), *True,* o *False*.

4.2.1.6 | El camp *Inicialitzacions* ha d'estar desactivat si la categoria de la qüestió no permet aquesta característica.

4.2.1.7 | El camp *Entrada* ha d'estar desactivat si la categoria de la qüestió no permet aquesta característica.

4.2.1.8 | El camp *Neteja* ha d'estar desactivat si la categoria de la qüestió no permet aquesta característica.

4.2.1.9 | El camp *Sortida* no s'ha de poder editar.

4.2.2.10 | S'ha de poder accedir a les comprovacions del joc de proves

4.2.3 Consulta de qüestions – Consulta de comprovacions

4.2.3.1 | Ha d'existir un llistat de totes les comprovacions del jocs de proves.

4.2.3.2 | S'ha de poder seleccionar una comprovació del llistat per consultar-ne les seves dades.

4.2.3.3 | Les dades que s'han de mostrar al consultar una comprovació:

Dades generals

- Nom (text)
- Descripció (textarea)
- Missatge d'error (textarea)
- Correcció binària (select)
- Consumeix intent (select)

Sentències SQL

- Entrada (textarea)
- Sortida (textarea)

4.2.3.4 | El camp *Missatge Error* ha de permetre introduïr contingut en 3 idiomes diferents (Català, Castella i Anglès).

4.2.3.5 | Els camps *Correcció binària* i *Consumeix intent* poden ser *NULL* (per defecte), *True,* o *False*.

4.2.3.6 | El camp *Sortida* no s'ha de poder editar.

4.2.4 Consulta de qüestions – Consulta de pesos

4.2.4.1 | Ha d'existir un llistat que mostri els pesos de cadascun dels jocs de proves de la qüestió, i que permeti modificar-los.

4.2.4.2 | Ha d'existir un indicador que indiqui si la suma dels pesos del tots jocs de proves és o no és igual a 1.

4.3. Modificació de qüestions

4.3.1 Modificació de qüestions – General

4.3.1.1 | S'ha de poder modificar les dades de la qüestió i confirmar-les.

4.3.1.2 | Al confirmar les modificacions, s'ha de preguntar a l'usuari si n'està segur i, alhora, presentar un resum amb canvis que s'han fet.

4.3.1.3 | Al confirmar les modificacions s'ha d'aturar el procés i notificar si no es compleixen les validacions:

- *Títol*: no pot ser buit.
- *Enunciat*: no pot ser buit en Català.
- *Dificultat:* ha d'estar entre 0 i 1.
- *Autor:* no pot ser buit.
- *Categoria:* no pot ser buit.
- *Esquema:* no pot ser buit.
- *Solució*: no pot ser buit, en cap de les seves múltiples formes.
- *Post-inicialitzacions JP*: no pot ser més gran que el número total de jocs de proves que té la qüestió.

4.3.1.4 | Les modificacions confirmades han d'acabar persistint correctament a la base de dades.

4.3.1.5 | Si *Tipus solució* no és *URL+usuario+clave*, els camps *URL*, *Usuari*, *Clau* i *SSL* s'han de posar a `NULL` a la base de dades quan es persisteix la qüestió.

4.3.1.6 | Si s'ha modificat algun dels camps *Inicialitzacions*, *Solució*, *Neteja*, *Post-inicialitzacions*, o *Post-inicialitzacions JP*, quan la qüestió es persisteix:

- S'ha de marcar com a *No resolta*.
- S'ha de netejar els camps *Sortida* de tots els jocs de proves i de totes les comprovacions de la qüestió, posant-los a `NULL`.

4.3.1.5 | S'han de poder tirar enrere els canvis que s'han fet si encara no s'han confirmat.

4.3.2 Modificació de qüestions – Modificació de jocs de proves

4.3.2.1 | S'ha de poder modificar les dades de cada joc de proves i confirmar-les.

4.3.2.2 | Al confirmar les modificacions, s'ha de preguntar a l'usuari si n'està segur i, alhora, presentar un resum amb canvis que s'han fet.

4.3.2.3 | Al confirmar les modificacions s'ha d'aturar el procés i notificar si no es compleixen per algun dels jocs de proves les validacions:

- *Nom*:
  - no pot ser buit;
  - no pot existir cap altre joc de proves a la qüestió amb el mateix valor.
- *Descripció*: no pot ser buit.
- *Missatge d'error*: no pot ser buit en Català.

4.3.2.4 | Les modificacions confirmades han d'acabar persistint correctament a la base de dades.

4.3.2.5 | Si s'ha modificat algun dels camps *Inicialitzacions*, *Entrada*, o *Neteja* d'algun joc de proves, o s'ha afegit o eliminat algun joc de proves, quan la qüestió es persisteix:

- La qüestió s'ha de marcar com a *No resolta*.
- S'ha de netejar els camps *Sortida* de tots els jocs de proves i de totes les comprovacions de la qüestió, posant-los a `NULL`.

4.3.2.6 | S'han de poder tirar enrere els canvis que s'han fet si encara no s'han confirmat.

4.3.2.7 | S'ha de poder afegir nous jocs de proves.

4.3.2.8 | S'ha de poder duplicar jocs de proves.

4.3.2.9 | S'ha de poder esborrar jocs de proves d'un en un.

- Quan es persisteix l'ordre d'esborrar algun joc de proces, també s'han d'esborrar totes les seves comprovacions de la base de dades.

4.3.3 Modificació de qüestions – Modificació de comprovacions

4.3.3.1 | S'ha de poder modificar les dades de cada comprovació i confirmar-les.

4.3.3.2 | Al confirmar les modificacions, s'ha de preguntar a l'usuari si n'està segur i, alhora, presentar un resum amb canvis que s'han fet.

4.3.3.3 | Al confirmar les modificacions s'ha d'aturar el procés i notificar si no es compleixen per alguna de les comprovacions les validacions:

- *Nom*:
  - no pot ser buit;
  - no pot existir cap altra comprovació al joc de proves amb el mateix valor.
- *Descripció*: no pot ser buit.
- *Missatge d'error*: no pot ser buit en Català.
- *Entrada*: no pot ser buit.

4.3.3.4 | Les modificacions confirmades han d'acabar persistint correctament a la base de dades.

4.3.3.5 | Si s'ha modificat algun dels camps *Entrada* d'alguna de les comprovacions, o s'ha afegit o eliminat alguna comprovació, quan la qüestió es persisteix:

- La qüestió s'ha de marcar com a *No resolta*.
- S'ha de netejar els camps *Sortida* de tots els jocs de proves i de totes les comprovacions de la qüestió, posant-los a `NULL`.

4.3.3.6 | S'han de poder tirar enrere els canvis que s'han fet si encara no s'han confirmat.

4.3.3.7 | S'ha de poder afegir noves comprovacions.

4.3.3.8 | S'ha de poder duplicar comprovacions.

4.3.3.9 | S'ha de poder esborrar comprovacions d'una en una.

4.3.4 Modificació de qüestions – Modificació de pesos

4.3.4.1 | S'ha de poder modificar els pesos de cada joc de proves.

4.3.4.2 | Al confirmar les modificacions, s'ha de preguntar a l'usuari si n'està segur i, alhora, presentar un resum amb canvis que s'han fet.

4.3.4.3 | Al confirmar les modificacions s'ha d'aturar el procés i notificar si no es compleixen per alguna de les comprovacions les validacions:

- *Pes*: ha d'estar entre 0 i 1.

4.3.4.4 | Les modificacions confirmades han d'acabar persistint correctament a la base de dades.

4.3.4.6 | S'han de poder tirar enrere els canvis que s'han fet si encara no s'han confirmat.

4.4. Alta, Baixa i Còpia de qüestions

4.4.1 | S'ha de poder afegir noves qüestions.

4.4.2 | S'ha de poder esborrar qüestions d'una en una.

- També s'han d'esborrar tots els seus jocs de proves i totes les seves comprovacions de la base de dades.

4.4.3 | S'ha de poder crear una nova qüestió a partir d'una qüestió existent.

5. Gestió categories

5.1 | Ha d'existir un llistat de totes les categories.

5.2 | S'ha de poder seleccionar una categoria del llistat per consultar-ne les seves dades:

- Identificador (text)
- Descripció (text)
- Característiques de qüestió (checkboxes)

Característiques de qüestió indica quins elements, d'entre tots els que pot tenir una qüestió, tenen sentit i quins no, per a totes les qüestions que són de la categoria. Són les següents:

- Inicialitzacions
- Post-inicialitzacions
- Neteja
- Estat BD
- Solució
- Inicialitzacions JP
- Entrada JP
- Neteja JP
- Comprovacions

5.3 | S'ha de poder modificar les dades de la categoria i confirmar-les.

5.4 | Al confirmar les modificacions s'ha d'aturar el procés i notificar si no es compleixen les validacions:

- *Identificador*:
  - ha de ser numèric;
  - no ha d'existir cap altre categoria amb aquest valor.

5.5 | Les modificacions confirmades han d'acabar persistint correctament a la base de dades.

5.6 | S'ha de poder tirar enrere els canvis que s'han fet si encara no s'han confirmat.

5.7 | Els canvis en les *Característiques de qüestió*, un cop confirmats, s'han de veure reflectits en la consulta de les qüestions que són de la categoria.

5.8 | S'ha de poder afegir noves categories.

5.9 | S'ha de poder esborrar categories d'una en una.

- S'ha de notificar un error i aturar el procés si: hi ha alguna entitat relacionada amb ella (qüestió o temàtica)

6. Gestió temàtiques

6.1 | Ha d'existir un llistat de totes les temàtiques.

6.2 | S'ha de poder seleccionar una temàtica del llistat per consultar-ne les seves dades:

- Nom (text)
- Descripció (textarea)
- Categories relacionades (select múltiple)

6.3 | Al camp *Categories relacionades* s'hi ofereixen per escollir totes les categories del sistema.

6.4 | S'ha de poder modificar les dades de la temàtica i confirmar-les.

6.5 | Al confirmar les modificacions s'ha d'aturar el procés i notificar si no es compleixen les validacions:

- *Nom*: no ha d'existir cap altre temàtica amb aquest valor.

6.6 | Les modificacions confirmades han d'acabar persistint correctament a la base de dades.

6.7 | S'ha de poder tirar enrere els canvis que s'han fet si encara no s'han confirmat.

6.8 | Els canvis en les *Categories relacionades* s'han de veure reflectits en el seleccionable de temàtiques que es mostra durant la consulta de les qüestions que són de la categoria.

6.9 | S'ha de poder afegir noves temàtiques.

6.10 | S'ha de poden esborrar temàtiques d'una en una.

- S'ha de notificar un error i aturar el procés si: hi ha alguna entitat relacionada amb ella (qüestió o categoria)

7. Gestió d'esquemes

7.1 | Ha d'existir un llistat de tots els esquemes.

7.2 | S'ha de poder seleccionar un esquema del llistat per consultar-ne les seves dades:

- Nom (text)
- Descripció (textarea)

7.3 | S'ha de poder modificar les dades de l'esquema i confirmar-les.

7.4 | Al confirmar les modificacions s'ha d'aturar el procés i notificar si no es compleixen les validacions:

- *Nom*: no ha d'existir cap altre esquema amb aquest valor

7.5 | Les modificacions confirmades han d'acabar persistint correctament a la base de dades.

7.6 | S'han de poder tirar enrere els canvis que s'han fet si encara no s'han confirmat.

7.7 | S'ha de poder afegir nous esquemes.

7.8 | S'ha de poder esborrar esquemes d'un en un.

  - S'ha de notificar un error i aturar el procés si: hi ha alguna qüestió relacionada amb ell.

Capítol 4: Arquitectura i solució

En aquest capítol s'adreça l'estat actual de la plataforma web i la visió que aquest projecte té per l'*Authoring Tool Web*, per després argumentar les tecnologies escollides per a dur-lo a terme.

4.1 Estat actual de la plataforma web

La plataforma web estàndard no sempre ha estat un ecosistema ideal per al desplegament d'aplicacions. De fet, no és fins fa relativament poc que ho comença a ser, després d'un procés lent de renovació.

Web tradicional

En el seus inicis, la web va ser dissenyada per servir només documents estàtics, enllaçats entre si. Una petició a un d'aquests document obtindria com a resposta una pàgina estàtica individual provinent del servidor en format *HTML*. Tot i que la posterior inclusió de formularis permetria a l'usuari enviar dades al servidor, qualsevol interacció encara requeriria substituir completament la pàgina actual per una de nova que reflectís el nou estat (*e.g.* mostrar un missatge com "les dades s'han enviat correctament"), perdent tot l'estat de la interfície d'usuari anterior.

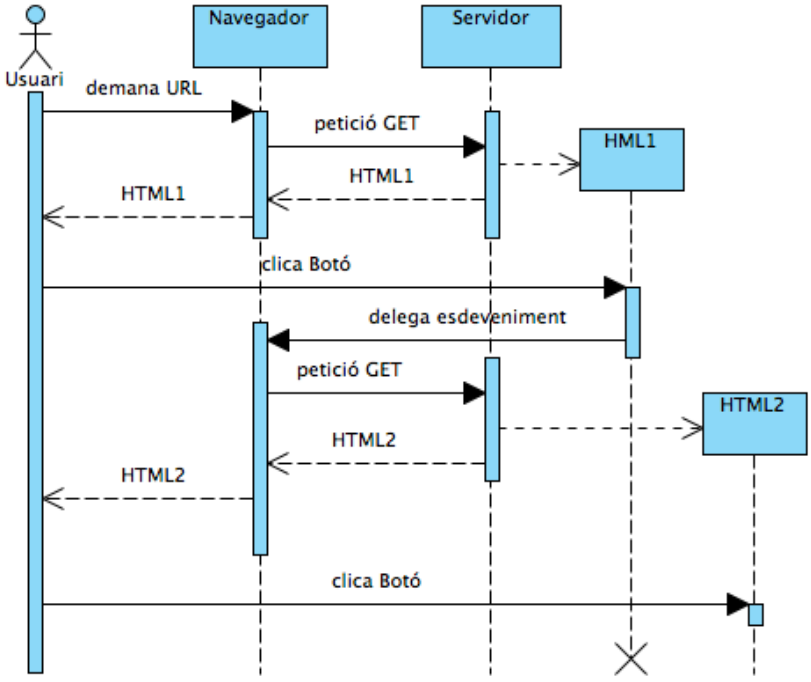


Figura 4.1 Flux d'una web tradicional

Web dinàmica i aplicacions web de client

Amb la creació i inclusió del llenguatge d'*scripting* de client anomenat *Javascript* per part de *Netscape* en el seu navegador el 1995, programadors podrien afegir elements dinàmics a les pàgines estàtiques servides. Això obriria un immens ventall de possibilitats, des d'acceptar *input* de l'usuari i mostrar resultats de càlculs immediatament, a modificar parts de la pàgina en funció de les accions de l'usuari, sense perdre l'estat general de la interfície. No obstant, la petició de nova informació, o l'enviament de dades al servidor encara requeriria el refresc total de la pàgina actual, perdent tot l'estat de la interfície un cop més. Es podria dir que acabaven de néixer les aplicacions web purament de client, com pot ser una calculadora.

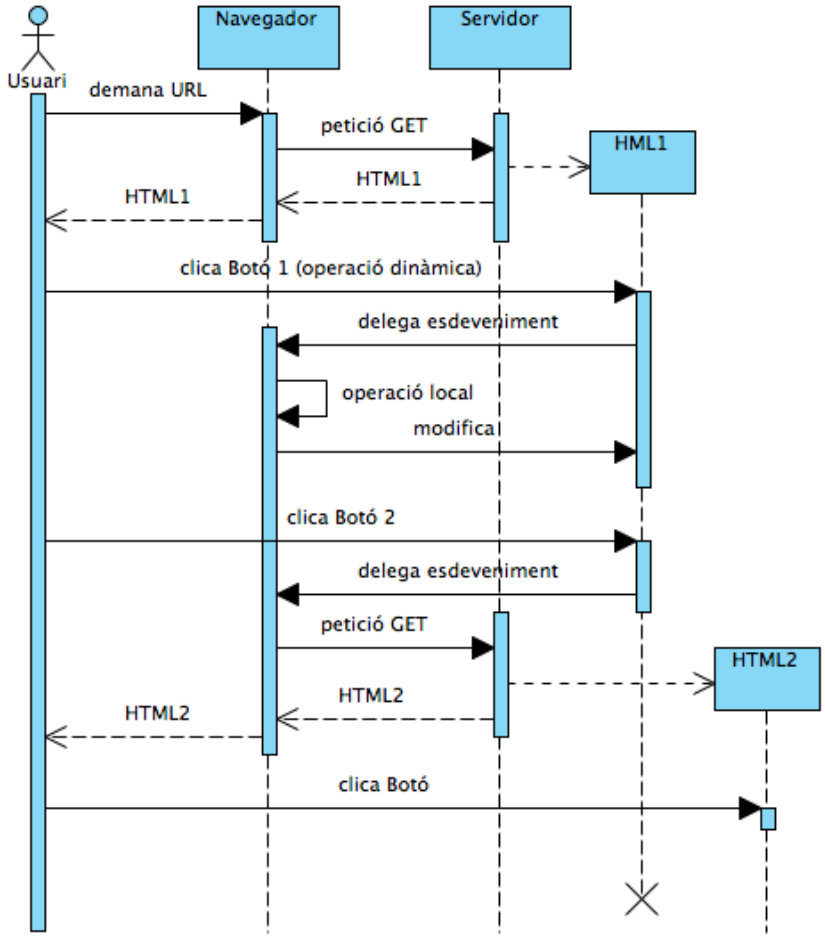


Figura 4.2 Flux d'una web dinàmica

Aplicacions web client-servidor, SPA

El 2005, es consolida l'especificació de l'objecte *XMLHttpRequest*[3] de *Javascript*, i tots els navegadors populars l'incorporen, donant lloc a la pràctica coneguda amb el nom d'*AJAX* (*Asynchronous Javascript And XML*). Aquest objecte brinda la funcionalitat necessària als programadors per fer peticions des del client cap al servidor en segon pla, i, amb les eines que ja coneixíem, modificar l'estat de la pàgina actual dinàmicament segons la resposta, sense necessitat de refrescar tota la pàgina.

És en aquest mateix moment quan la plataforma web reuneix les condicions necessàries per a desplegar-hi aplicacions *client-servidor* natives. El servidor serveix al navegador l'equivalent al software client en la primera petició que es fa, i aquest només fa peticions al servidor addicionals en segon pla per intercanviar informació o demanar recursos necessaris. El software client que es manté al navegador és capaç de processar les respostes del servidor i actualitzar selectivament l'estat de la interfície d'usuari dinàmicament sense necessitat de perdre'n tota la resta. Aquest tipus d'aplicacions també es coneix amb el nom de *SPA*[4] (*Single Page Application*). Un bon exemple d'aquest tipus d'aplicacions és el client web de correu *GMail* de *Google.*

---

[3] *XMLHttpRequest* (https://en.wikipedia.org/wiki/XMLHttpRequest)

[4] *SPA - Single Page Application* (https://en.wikipedia.org/wiki/Single-page_application)

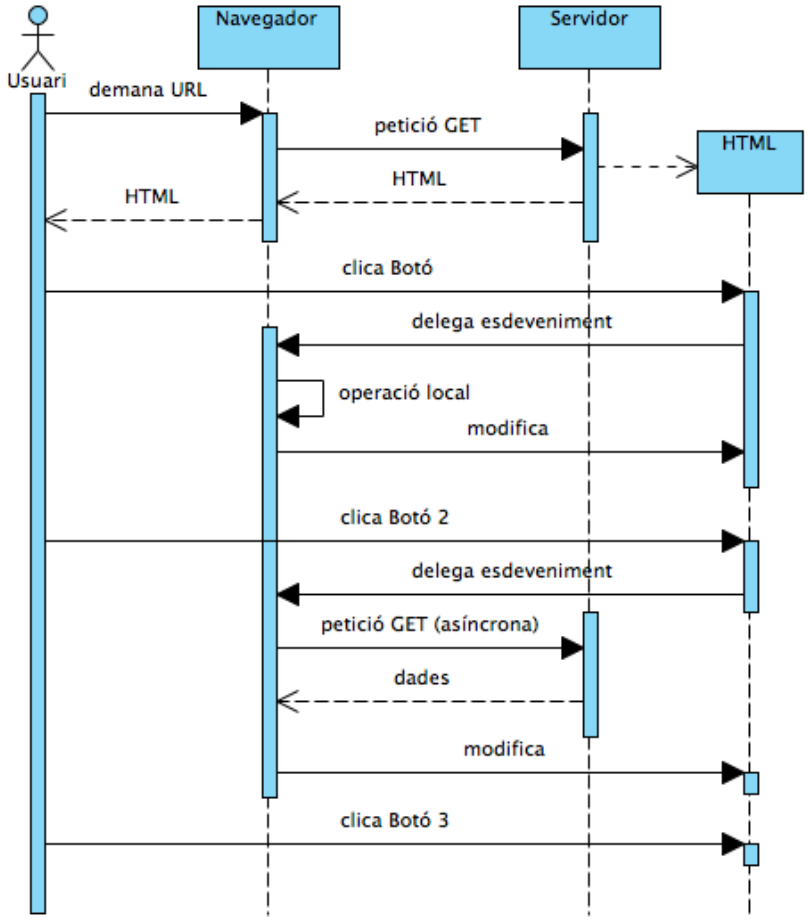


Figura 4.3 Flux d'una SPA

HTML5

Tot i això, els estàndards encara no contemplaven molts dels casos d'ús que aquest tipus d'aplicacions requeririen, i no és fins el 2011 quan, dins del marc de funcionalitats *HTML5*, es regulen i es comencen a implementar per part dels navegadors una sèrie de noves funcionalitats, que permeten al navegador, entre d'altres, manegar fitxers (*File API*), canviar la URL programàticament (*History API*), accedir a la localització de l'usuari (*Geolocation API*), reproduir àudio i vídeo sense necessitat de solucions externes, emmagatzemar dades entre sessions (*Storage API*), o obrir sockets de comunicació amb el servidor (*WebSockets API*), així com nous components com per exemple *inputs* del tipus data o del tipus numèric.

Aparença i posicionament d'elements

En quant a aparença gràfica i posicionament d'elements, la plataforma web compta amb l'especificació d'estils *CSS (Cascading Style Sheet)*. Mitjançant una microsintaxi de selectors, podem assignar declarativament propietats gràfiques a elements que existeixin dins del document web, i el navegador s'encarrega de pintar-los correctament. No obstant, es pot dir que aquesta especificació no ha evolucionat mai prou per afavorir plenament el posicionament d'elements com s'esperaria d'una aplicació real. Posicionaments tan simples com el de la figura següent no han estat mai possibles sense la necessitat d'ajuda externa (*Javascript*), tenint en compte que l'espai disponible és variable en funció de la mida de la finestra de l'usuari, o la resolució de la seva pantalla:

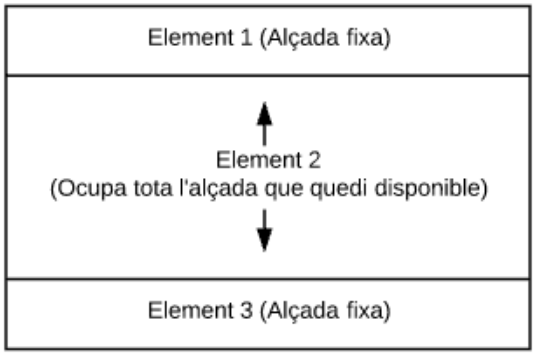


Figura 4.4

Per sort, actualment, encara en període experimental, s'està implementant una nova especificació dins del marc del *CSS* anomenada *Flexbox*[5], que permet, entre molts d'altres, el posicionament plantejat anteriorment de manera totalment nativa, i, en general, un posicionament molt més flexible i dinàmic d'elements. Aquest tipus de posicionament, sens dubte, acosta les interfícies gràfiques de la plataforma web al que estem acostumats a veure en aplicacions d'escriptori convencionals.

[5] *Flexbox* (https://www.w3.org/TR/css-flexbox-1/)

4.2 *Single Page Application* (*SPA*)

És, doncs, en el marc de les *SPA* on volem situar l'*Authoring Tool Web* que desenvolupem en aquest projecte, al caracteritzar-se principalment per oferir una experiència d'usuari fluida equiparable a la que ofereixen les aplicacions d'escriptori.

Per més claredat, es llisten els avantatges de les *SPA* respecte la web tradicional a continuació:

- Reactivitat de la interfície gràfica.
- Preservació de l'estat anterior.

Pot semblar una tonteria, però cal deixar clar que, quan parlem d'una aplicació client-servidor en general, en realitat estem parlant de dues aplicacions diferents:

1. L'aplicació client, encarregada de presentar informació a l'usuari, i reaccionar al seu *input*.
2. L'aplicació servidor, encarregada de la lògica de negoci del sistema. Aquesta ha de posar a disposició de l'aplicació client un contracte d'operacions per a què aquesta hi pugui interaccionar quan ho necessiti (serveis web).

Aquestes dues aplicacions comparteixen un objectiu, i estan connectades entre si per un canal de comunicació comú.

En el cas de les *SPA*, l'aplicació client ha d'estar desenvolupada en els llenguatges que entén el navegador: *HTML*, *CSS* i *Javascript*, i l'aplicació servidor en qualsevol dels llenguatges de programació acceptats pel gran ventall de servidors web que existeixen: JAVA, PHP, Ruby, Python, etc. La comunicació es fa usant el protocol HTTP(s) a través de la xarxa.

Així doncs, l'*Authoring Tool Web* necessitarà del desenvolupament d'aquestes dues aplicacions.

Figura 4.5 Arquitectura d'una SPA

4.3 Tecnologies escollides

4.3.1 Aplicació servidor (serveis web)

La pràctica més comuna actualment en el desenvolupament d'aplicacions servidor en general és usar el patró de disseny *Front-Controller*[6]. Aquest patró imposa un únic punt d'entrada per a totes les peticions externes que rep l'aplicació, cosa que permet unificar tots els processos relacionats amb la petició. Aquest únic punt d'entrada s'ha d'encarregar de:

1. Processar les dades de la petició.
2. Identificar el tipus de petició i delegar-la al component apropiat perquè la resolgui adequadament.
3. Respondre la petició.

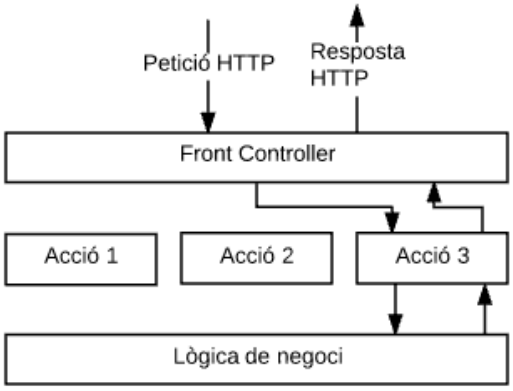


Figura 4.6 Patró Front-Controller

La majoria de *frameworks* populars per estructurar aplicacions servidor implementen aquest patró. Com que un dels objectius del projecte és re-aprofitar la major part de lògica de negoci de l'aplicació d'escriptori existent, i aquesta està codificada en *JAVA*, s'ha triat usar *Struts2*[7], al ser un *framework JAVA* que implementa el patró *Front Controller* i que no imposa restriccions en com estructurar la lògica de negoci.

---

[6] *Front-Controller* (https://en.wikipedia.org/wiki/Front_controller)

[7] *Struts2* (https://struts.apache.org)

4.3.2 Aplicació client

L'aplicació client és una mica més complicada. En contraposició a una aplicació servidor, que només ha de retornar una "fotografia" del model de l'aplicació en un moment concret en el temps, l'aplicació client ha d'anar maquillant dinàmicament l'única "fotografia" que té, a mesura que el model va canviant de manera constant. Aquesta és una tasca feixuga de controlar per part del desenvolupador, i sovint resulta en una barbaritat de codi font, ja que un sol canvi en el model pot comportar haver de tenir en compte que s'han de modificar una gran quantitat de components de la interfície gràfica. Per regular aquest paradigma, existeix el patró de disseny *Model-Vista-Controlador* (*MVC*)[8]:

1. Vista:
    a. S'encarrega de representar gràficament l'estat del model.
    b. S'encarrega d'acceptar les accions de l'usuari.
2. Controlador: S'encarrega de donar sentit a les accions de l'usuari, i les tradueix en operacions cap al model.
3. Model:
    a. S'encarrega de mantenir l'estat de la vista.
    b. Quan hi ha un canvi, ha d'existir un mecanisme que notifiqui automàticament a la vista el nou model de dades per a que s'actualitzi degudament.

[8] Model-Vista-Controlador (MVC) (https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller)

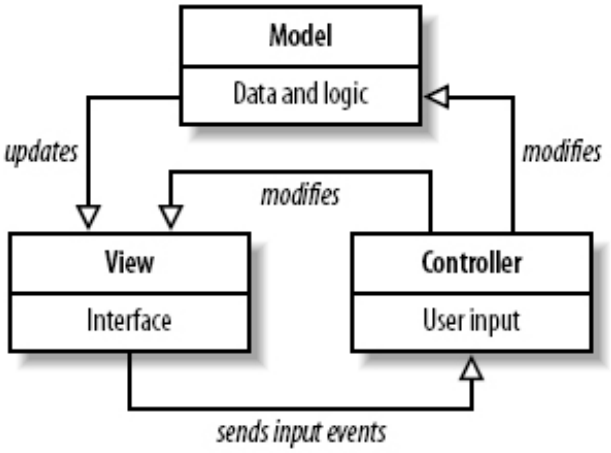


Figura 4.7 Patró Model-View-Controller[9]

Tot i que aquest no és un concepte nou, la vida dels *frameworks* de client (*Javascript*) que l'implementen per facilitar la vida al desenvolupador és més aviat jove. L'insuficient rendiment de l'intèrpret de *Javascript* i del motor de *render* dels navegadors n'ha estat una limitació durant molt de temps. No obstant, en el moment de desenvolupar aquest projecte ja existeixen opcions amb prou adopció, i prou demanda laboral, com per considerar-les. Les més populars: *Backbone*, *Ember* o *AngularJS* (i en el moment d'escriure'l, *React* i *Angular2*).

[9] Font: http://www.moock.org/lectures/mvc/

Amb la finalitat de desenvolupar una aplicació molt visual, reactiva, i molt interactiva, d'una manera àgil, i també a nivell d'interès personal, s'ha volgut, efectivament, explorar un d'aquests *frameworks MVC* de client per desenvolupar l'aplicació client del projecte. Després d'un període d'estudi, s'ha escollit *AngularJS*[10] pels següents motius:

- Promou l'organització de la capa de presentació en components fets a mida.
- Promou l'*HTML* com a element per donar estructura a l'arbre de components de l'aplicació.
- Promou l'enfoc declaratiu en el procés de definició d'una vista, utilitzant senzillament *HTML* enriquit, enlloc de l'enfoc programàtic, com fan altres frameworks com *Google Web Toolkit*[11].
- Promou la no-manipulació manual del *DOM*[12] per part del desenvolupador, en favor de només haver de manipular el model de dades. El framework s'encarrega d'actualitzar el *DOM* adequadament. Un dels problemes més grans en la història de les aplicacions web de client.
- Ofereix *two-way data-binding*: si el model s'actualitza, la vista s'actualitza, i si l'usuari actualitza la vista (mitjançant algun camp de formulari, per exemple), el model s'actualitza.
- I tot això ho fa sense imposar estructures de dades pròpies com fa, per exemple, *EmberJS*[13]. A AngularJS es treballa amb *POJO*s (*Plain Old Javascript Objects*) o objectes simples de *Javascript*.

En quant a l'aspecte visual, s'ha decidit incloure *Flexbox*[14] en el projecte, la nova especificació dins del marc de *CSS*, que, tot i encara no estar acabada, ja és prou funcional en els navegadors més populars com per ajudar-nos a construir una interfície d'usuari similar a la d'una aplicació d'escriptori.

---

[10] *AngularJS* (https://angularjs.org/)

[11] *GWT - Google Web Toolkit* (http://gwtproject.org)

[12] *DOM – Document Object Model* (https://www.w3.org/DOM/)

[13] *EmberJS* (http://emberjs.com/)

[14] *Flexbox* (https://www.w3.org/TR/css-flexbox-1/)

4.3.3 Transport de dades

Per a la comunicació de dades entre el client i el servidor s'ha escollit utilitzar el format d'intercanvi *JSON*[15] (*JavaScript Object Notation*), per la seva lleugeresa, facilitat de maneig i compatibilitat amb altres projectes actuals.

[15] JSON (http://www.json.org/)

4.4 Resum

Es vol que l'*Authoring Tool Web* sigui una *Single Page Application* per garantir una experiència d'usuari similar a la d'una aplicació d'escriptori:

- Reaccionant immediatament a les accions de l'usuari, donant *feedback* immediat a aquestes accions.
- Mantenint sempre l'estat anterior.

Una *SPA* consisteix de dues aplicacions.

L'aplicació servidor serà un compendi de serveis webs i es desenvoluparà en *JAVA*, utilitzant el *framework Struts2*, beneficiant-nos de la seva implementació del *Front-Controller* per a processar les peticions, i utilitzant la lògica de negoci de l'*Authoring Tool* existent.

L'aplicació client serà una aplicació desenvolupada en els llenguatges natius del navegador (*HTML*, *Javascript*, *CSS*), explorant les possibilitats del *framework AngularJS*, beneficiant-nos de la seva implementació de l'*MVC* per aconseguir una alta reactivitat de la interfície gràfica sense una complexitat de codi font alta. En quant a l'aparença i distribució d'elements, s'explorarà també la nova especificació de *CSS* anomenada *Flexbox*, al permetre'ns distribuir elements gràfics d'una manera molt més pròxima a la d'una interfície d'aplicació d'escriptori.

El format d'intercanvi entre l'aplicació client i l'aplicació servidor (i viceversa) serà el *JSON*.

Aquesta arquitectura, a més a més, desacobla la tecnologia de l'aplicació client de la tecnologia de l'aplicació servidor, cosa que potencialment permet:

- Amagar la implementació interna de l'altra aplicació (per seguretat i també per abstracció) quan només s'ha de treballar amb una d'elles. (*e.g. Outsourcing* de l'aplicació de client).
- Substituir l'aplicació client amb facilitat quan sigui necessari (o la de servidor, tot i que menys probable), sempre que es respectin els contractes.
- Permetre l'accés a la lògica de negoci a través dels serveis web de l'aplicació servidor a altres aplicacions clients (mòbil, processos automatitzats, etc.)

Per últim, les tecnologies escollides per l'aplicació client treballen amb els llenguatges natius del navegador, i no amb abstraccions en altres llenguatges, afavorint:

- Tot el sector de desenvolupadors acostumat a treballar amb aquests.

- No tancar l'aplicació client només a desenvolupadors, i obrir-la a altres rols professionals també acostumats a treballar amb aquests llenguatges (dissenyadors-maquetadors, ui/ux experts, etc.).

Capítol 5: Disseny i implementació: Servidor

En el disseny i la implementació de l'aplicació servidor, s'ha treballat sobre el disseny i el codi de l'aplicació existent, al ser la majoria de plantejaments encara vàlids.

No obstant, de l'organització en 3 capes de l'aplicació existent, la capa de presentació de ja no té sentit, i per tant, en aquest capítol no se'n parla, ja que l'aplicació servidor no s'encarrega de presentar dades a l'usuari, sinó de respondre a peticions en forma de servei web.

L'aplicació servidor està estructurada en els següents paquets, que s'aniran tractant a continuació:

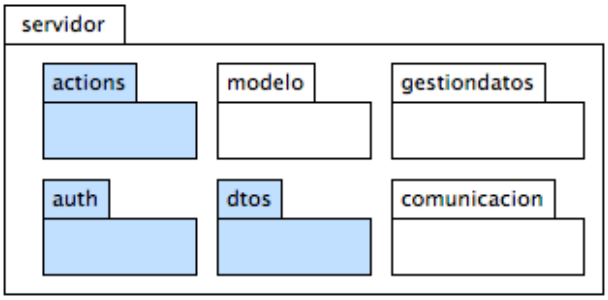


Figura 5.1 Paquets de l'aplicació servidor. En blau els que s'han afegit nous.

5.1 Punts d'entrada

Tot seguit s'especifiquen les *URL*s de tots els recursos de l'aplicació servidor, i la seva funcionalitat esperada.

URL	Mètode HTTP	Protegit	Cós	Resposta	Comentaris

`/`
`/questions*`
`/categories*`
`/thematics*`
`/schemas*`
	`GET`	sí		`HTML`	Serveix l'aplicació client

`/login`	`GET`	no		`HTML`	Mostra el formulari d'accés
`/login`	`POST`	no		*Redirect*	Processa una petició d'accés
`/logout`	`GET`	sí		*Redirect*	Finalitza la sessió
`/ws/questions`	`GET`	sí		`JSON`	Retorna el llistat de totes les qüestions
`/ws/questions`	`POST`	sí	Dades de la nova qüestió (JSON)	`JSON`	Crea una nova qüestió i la retorna.
`/ws/questions/{id}`	`PUT`	sí	Noves dades de la qüestió (JSON)	`JSON`	Modifica la qüestió {id} i la retorna.
`/ws/questions/{id}`	`DELETE`	sí		`JSON`	Esborra la qüestió {id}
`/ws/categories`	`GET`	sí		`JSON`	Retorna el llistat de totes les categories
`/ws/categories`	`POST`	sí	Dades de la nova categoria (JSON)	`JSON`	Crea una nova categoria i la retorna.
`/ws/categories/{id}`	`PUT`	sí	Noves dades de la qüestió (JSON)	`JSON`	Modifica la categoria {id} i la retorna.
`/ws/categories/{id}`	`DELETE`	sí		`JSON`	Esborra la categoria {id}
`/ws/thematics`	`GET`	sí		`JSON`	Retorna el llistat de totes les

					temàtiques
/ws/thematics	POST	sí	Dades de la nova temàtica (JSON)	JSON	Crea una nova temàtica i la retorna.
/ws/thematics/{id}	PUT	sí	Noves dades de la temàtica (JSON)	JSON	Modifica la temàtica {id} i la retorna.
/ws/thematics/{id}	DELETE	sí		JSON	Esborra la temàtica {id}
/ws/schemas	GET	sí		JSON	Retorna el llistat de tots els esquemes
/ws/shchemas	POST	sí	Dades del nou esquema (JSON)	JSON	Crea un nou esquema i el retorna
/ws/shcemas/{id}	PUT	sí	Noves dades de l'esquema (JSON)	JSON	Modifica l'esquema {id} i el retorna
/ws/schemas/{id}	DELETE	sí		JSON	Esborra l'esquema{id}
/ws/solutiontypes	GET	sí		JSON	Retorna el llistat de tots els tipus de solució

La majoria dels punts d'entrada es corresponen amb els principals casos d'ús del sistema, vistos a l'estudi funcional del *Capítol 3*.

Com es pot observar, per simplicitat del sistema s'ha decidit que tant les funcionalitats d'accés com la de servir l'aplicació client estiguin dins de l'aplicació servidor. Un cop l'aplicació client és servida, aquesta només es comunica amb els serveis web que hi ha sota el namespace `/ws`.

5.2 Actions

Les *Action*s són les classes encarregades d'atendre la necessitat concreta d'una petició *HTTP(s)* que arriba a l'aplicació servidor *Struts2*. El *framework* s'encarrega d'escollir l'*Action* que correspon, en funció de la URL de la petició, i de preparar correctament totes les dades que necessiti.

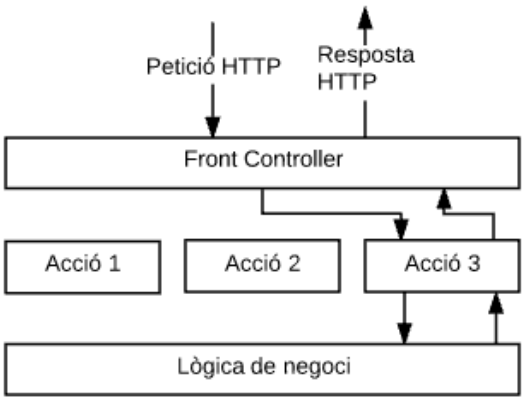


Figura 5.2

Les *Action*s es poden organitzar en paquets, i així assignar-los diferents tipus de pre-processos a cadascun, segons sigui convenient. Tota la configuració que afecta a les *Action*s es troba al fitxer `struts.xml` a l'arrel del projecte. En aquest fitxer s'hi poden observar els següents paquets d'*Action*s:

1. `public`: Conté les accions públiques que no requereixen autenticació de l'usuari.
2. `secured`: Conté les accions que requereixen l'autenticació de l'usuari, però que no formen part dels serveis web que consumeix l'aplicació client. Inclou el `DBConnectionInterceptor` a la seva pila de pre-procés per comprovar que l'usuari està autenticat abans de prosseguir (Veure secció 5.6).
3. `secured-ws`: Conté les accions que consumeix l'aplicació client (que totes requereixen autenticació). Inclou el `DBConnectionInterceptor`, i també el `JSONInterceptor` a la seva pila de pre-procés per acceptar *input* en format *JSON*.

A continuació veiem la relació entre *URL*s i les *Action*s del sistema:

URL	**Action**	**Package**

`/`
`/questions*`
`/categories*`
`/thematics*`
`/schemas*`
	Serveix directament l'aplicació client sense passar per cap acció.
	`/static/client/bin/index.html`
		`secured`

`/login`	`auth.LoginAction`	`public`
`/logout`	`auth.LogoutAction`	`secured`
`/ws/questions`	`questions.QuestionsAction`	`secured-ws`
`/ws/questions/{id}`	`questions.QuestionAction`	`secured-ws`
`/ws/categories`	`categories.CategoriesAction`	`secured-ws`
`/ws/categories/{id}`	`categories.CategoryAction`	`secured-ws`
`/ws/thematics`	`thematics.ThematicsAction`	`secured-ws`
`/ws/thematics/{id}`	`thematics.ThematicAction`	`secured-ws`
`/ws/schemas`	`schemas.SchemasAction`	`secured-ws`
`/ws/schemas/{id}`	`schemas.SchemaAction`	`secured-ws`
`/ws/solutiontypes`	`solutiontypes.SolutiontypesAction`	`secured-ws`

Com es pot veure, s'han agrupat les funcionalitats dels serveis webs segons si afecten a una sola entitat del sistema, o n'afecten la col·lecció. Per distingir de quina operació es tracta concretament, consultem el "mètode" de la petició *HTTP (GET, POST, PUT, DELETE)*. A continuació veiem com distingim una operació de modificació d'una operació d'esborrat a l'acció `CategoryAction`.

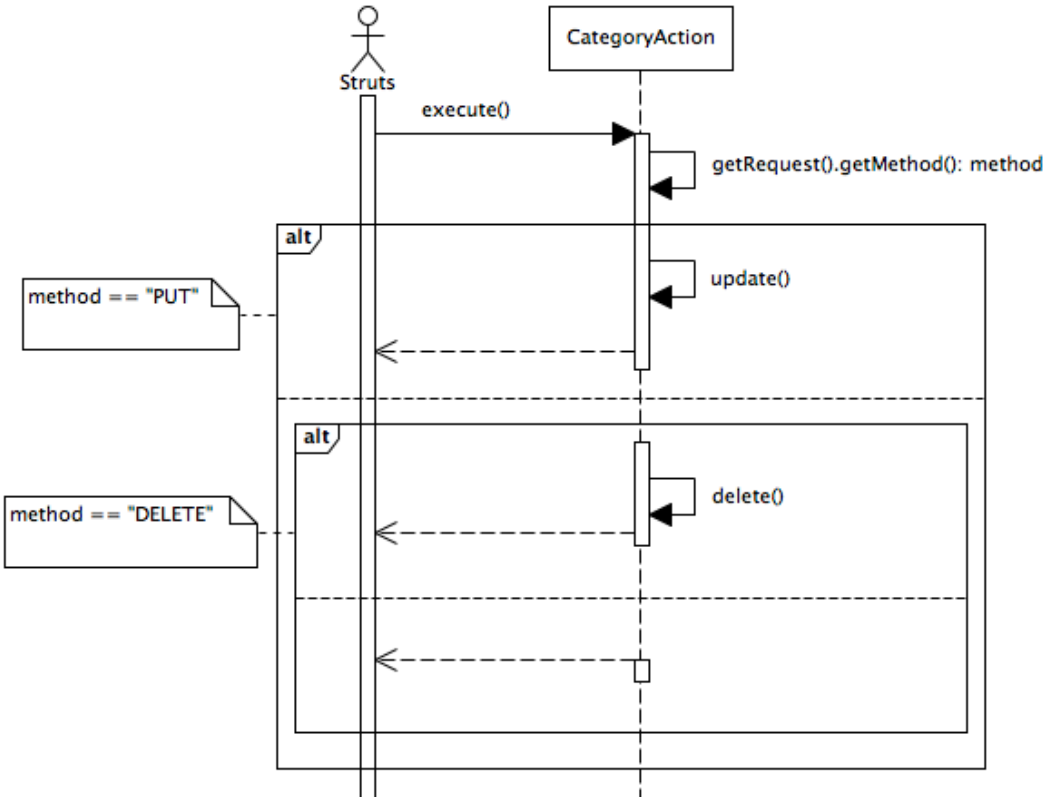

Struts
CategoryAction
execute()
getRequest().getMethod(): method
alt
method == "PUT"
update()
alt
method == "DELETE"
delete()

Figura 5.3

5.3 Accés a dades (*gestiondatos*)

Les dades que ha de gestionar la nova aplicació són les mateixes que les de l'aplicació existent. Així doncs, la capa de gestió de dades es pot aprofitar en la seva totalitat.

No obstant, s'hi ha hagut de fer alguna millora:

- S'ha donat a totes les classes *Ctrl* una manera d'obtenir un sol registre de la base de dades. Abans no era necessària perquè això es feia des dels diccionaris del domini, que ja tenien tots els registres de la base de dades pre-carregats, però els diccionaris han de desaparèixer, com veurem més endavant.
- S'ha eliminat el concepte de *Catalogo* i s'ha unit tota la seva funcionalitat amb el `CtrlCuestion`.
- S'ha optimitzat la manera en la que es consulten les dades (i sub-dades) d'una qüestió, i també la manera en la que es construeix l'entitat resultant.

5.4 Domini (*modelo*)

Per a la capa de domini s'aprofita la majoria del plantejament de l'aplicació existent.

Coses que no ha de fer la nova capa de domini respecte l'anterior:

- Gestionar l'idioma de l'aplicació. És responsabilitat de l'aplicació client.
- Mantenir llistats d'entitats en memòria (diccionaris). L'aplicació servidor no s'ha de preocupar de mantenir un estat per a la presentació d'informació. Aquesta responsabilitat es traspassa també a l'aplicació de client. (Recordem que l'aplicació de servidor es limita a retornar una fotografia de l'estat actual del model, no a fer-li un seguiment d'evolució).

Amb l'eliminació dels diccionaris (*Dicc*s), tenim la necessitat d'un lloc on agrupar totes les operacions relacionades amb una entitat del Model. Per aquest motiu introduïm les classes *Service*. Més concretament, la seva responsabilitat és:

1. Fer d'intermediari entre el consumidor i les operacions d'accés a dades.
2. Assegurar la validesa de les entitats que es té intenció de persistir. (Validació).
3. Serialitzar/Des-serialitzar entitats cap a / des de la seva versió de transport. (*DTO*).

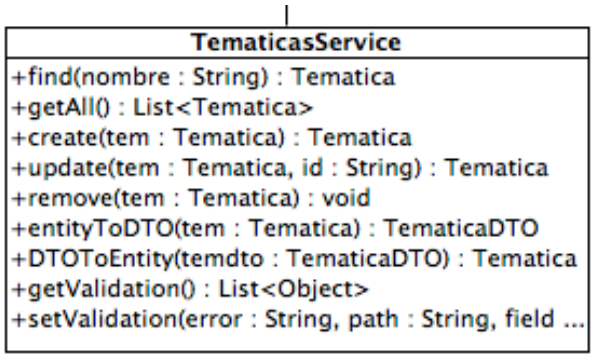


Figura 5.4 Exemple de Service

Moltes de les funcionalitats que ofereixen els *Service*s ja existien als *Dicc*s, així que les hem re-aprofitat i adaptat.

Així doncs, la capa de domini, en la implementació és el *package* `modelo`, quedaria organitzada de la següent manera:

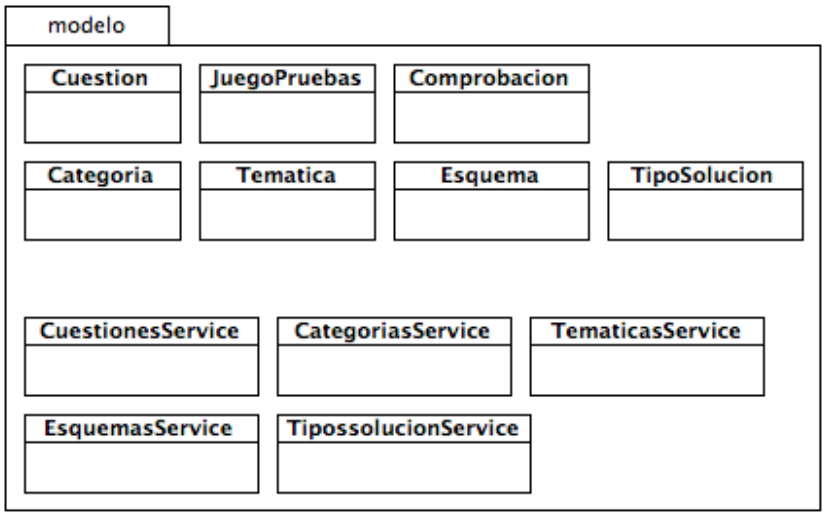


Figura 5.5

Veiem tot seguit un exemple d'interacció amb validació amb els *Service*s.

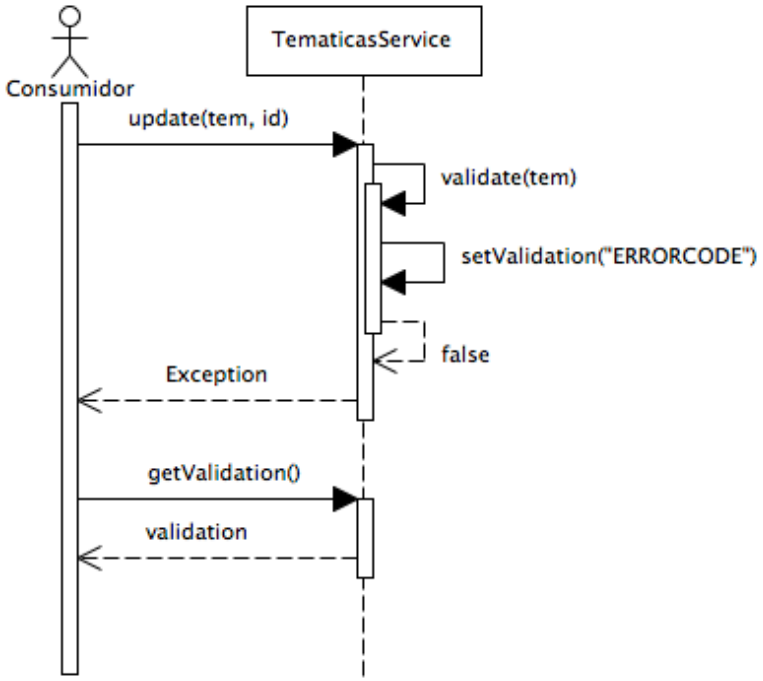


Figura 5.6 Modificació d'una temàtica. Cas: dades invàlides.

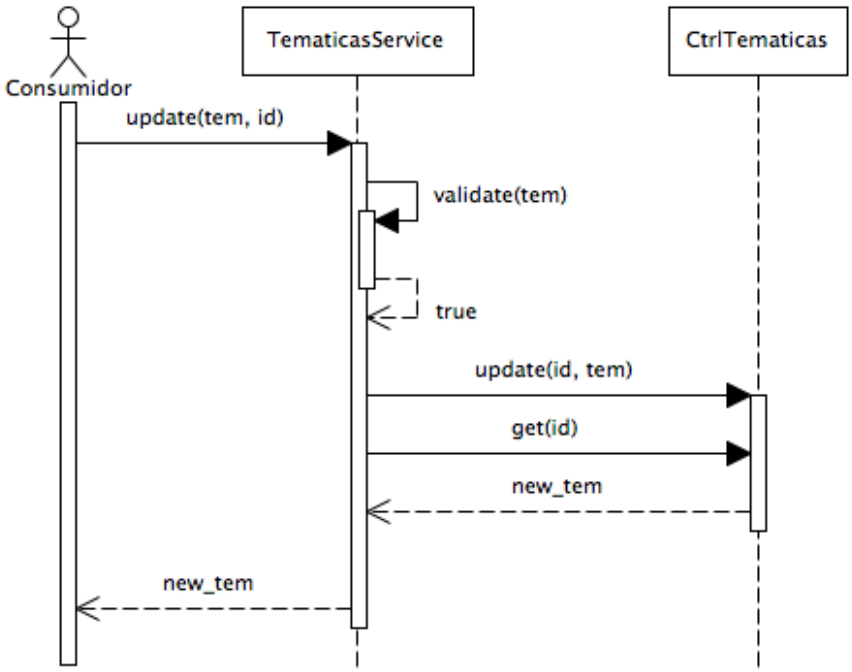


Figura 5.7 Modificació d'una temàtica. Cas: OK.

5.5 Intercanvi d'informació entre client i servidor *(DTO*s)

Els serveis web que consumeix l'aplicació client han de ser capaços de rebre i enviar dades. Per una banda, les dades que el client envia al servidor han de ser convertides en una entitat amb la qual l'aplicació servidor sàpiga treballar. Per l'altra, el servidor ha de ser capaç de convertir aquestes entitats amb les quals treballa a dades que el client pugui entendre.

Com que hem reaprofitat la capa de dades de l'aplicació existent, hem conservat també les entitats amb les que aquesta treballa. Aquestes entitats (sobretot `Cuestion`) tenen una estructura complexa, redundant i no normalitzada (sub-entitats), que faria la informació d'intercanvi massa pesada i difícil de manegar pel client. Per aquest motiu, per cada entitat, s'ha creat una classe *DTO* (*Data Transfer Object*) equivalent, que usem tant per enviar les dades d'una entitat a l'aplicació client, com per rebre-les.

Podem veure a continuació el *DTO* per l'entitat `Cuestion`.

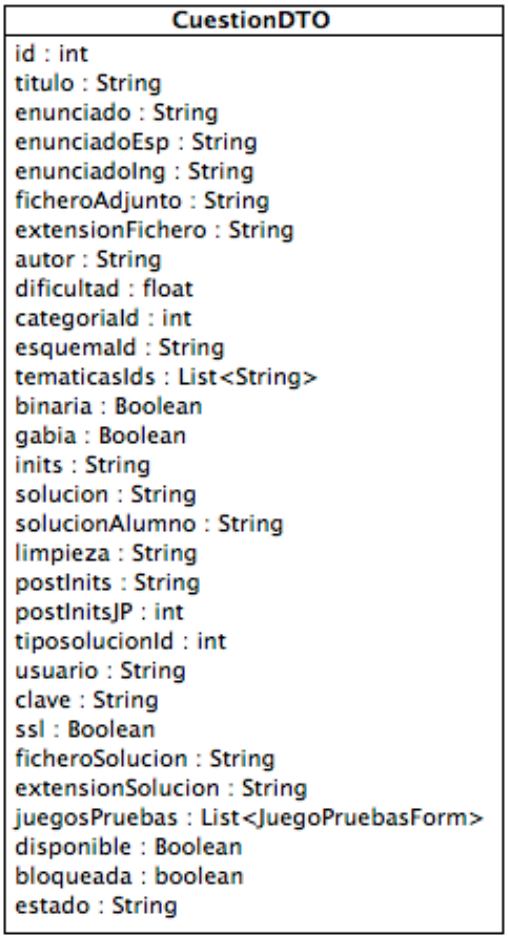


Figura 5.8

Com a norma general s'ha normalitzat totes les sub-entitats de les quals l'aplicació client pot aconseguir la seva informació completa usant el servei web de l'entitat corresponent. Pel cas de `Cuestion`, aquestes sub-entitats són la categoria, l'esquema, les temàtiques i el tipus de solució, com veiem amb els respectius camps `categoriaId`, `esquemaId`, `tematicasIds`, i `tiposolucionId`. Pel contrari, sub-entitats que no es poden consultar per cap altra banda no es normalitzen, però s'inclouen també en la seva versió *DTO*, com és el cas dels jocs de proves (`juegosPruebas`). En el cas dels fitxers (`ficheroAdjunto` i `ficheroSolucion`), només s'envien els seus continguts cap al client si són de tipus text. En cas contrari el client els haurà d'anar a buscar a una acció especial destinada a servir fitxers.

Quan es demana un servei web de consulta d'alguna entitat, l'aplicació servidor ha de retornar un llistat de *DTO*s d'aquella entitat. Per fer-ho, l'*Action* corresponent primer usa les funcionalitats de domini per obtenir les entitats demanades, i després les converteix en *DTO*, usant les funcionalitats que ens ofereixen els *Service*s de la capa de domini, per després retornar-les com a resultat en *JSON*.

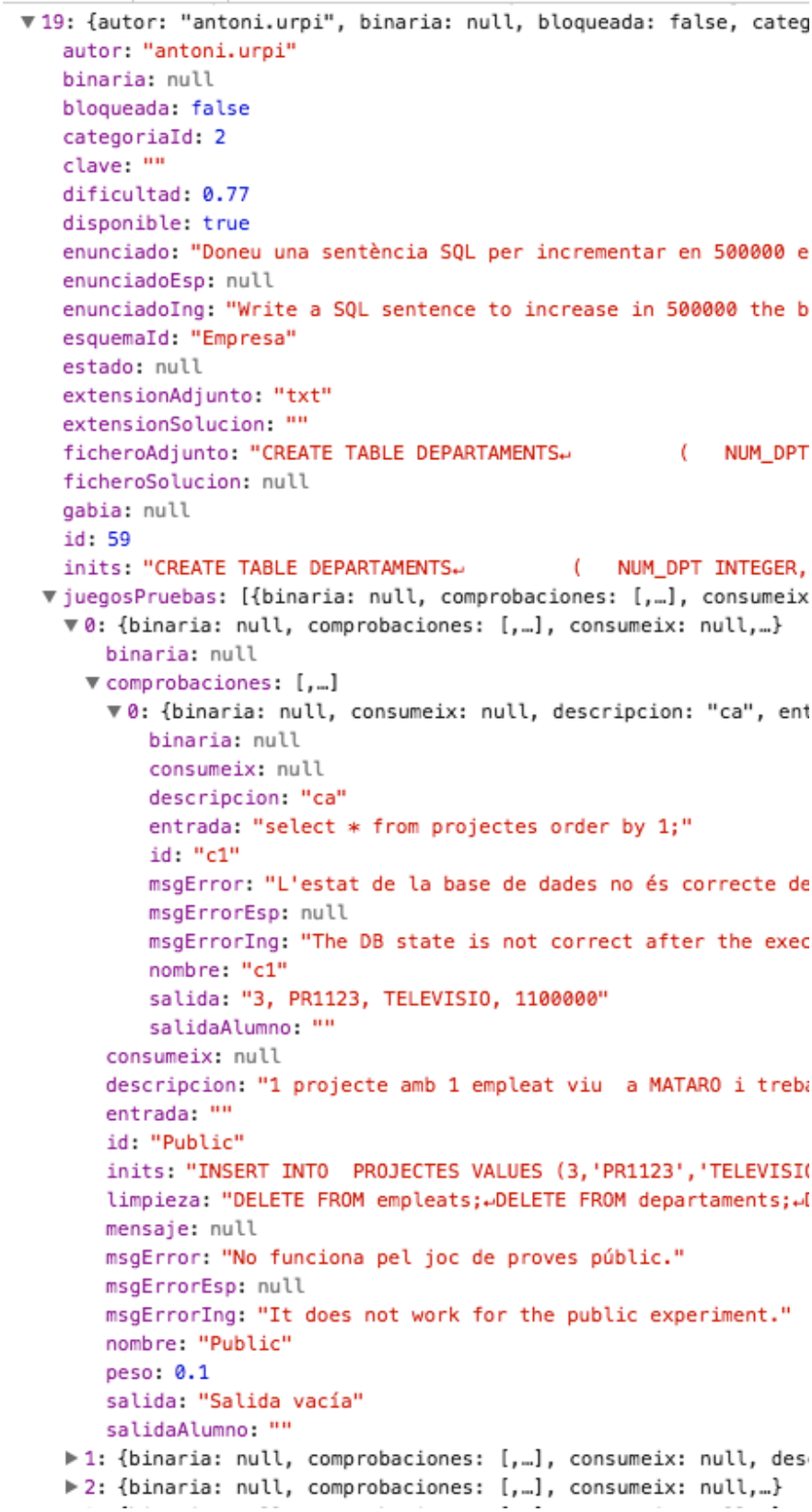


Figura 5.9 Extracte (incomplet) de l'objecte DTO rebut per l'aplicació client al navegador

Quan es demana un servei web de creació o modificació, l'aplicació client ha d'enviar informació respectant el contracte del *DTO* a l'aplicació servidor en el cós de la petició. Un cop la petició arriba al servidor, el `JSONInterceptor` s'encarrega de convertir aquest cós de la petició en l'objecte *DTO* corresponent, i deixar-lo disponible a l'*Action* per a ser usat. Un cop tenim el *DTO* podem convertir-lo a l'entitat original amb la que el sistema sap treballar mitjançant les funcionalitats que ens ofereixen els *Service*s de la capa de domini.

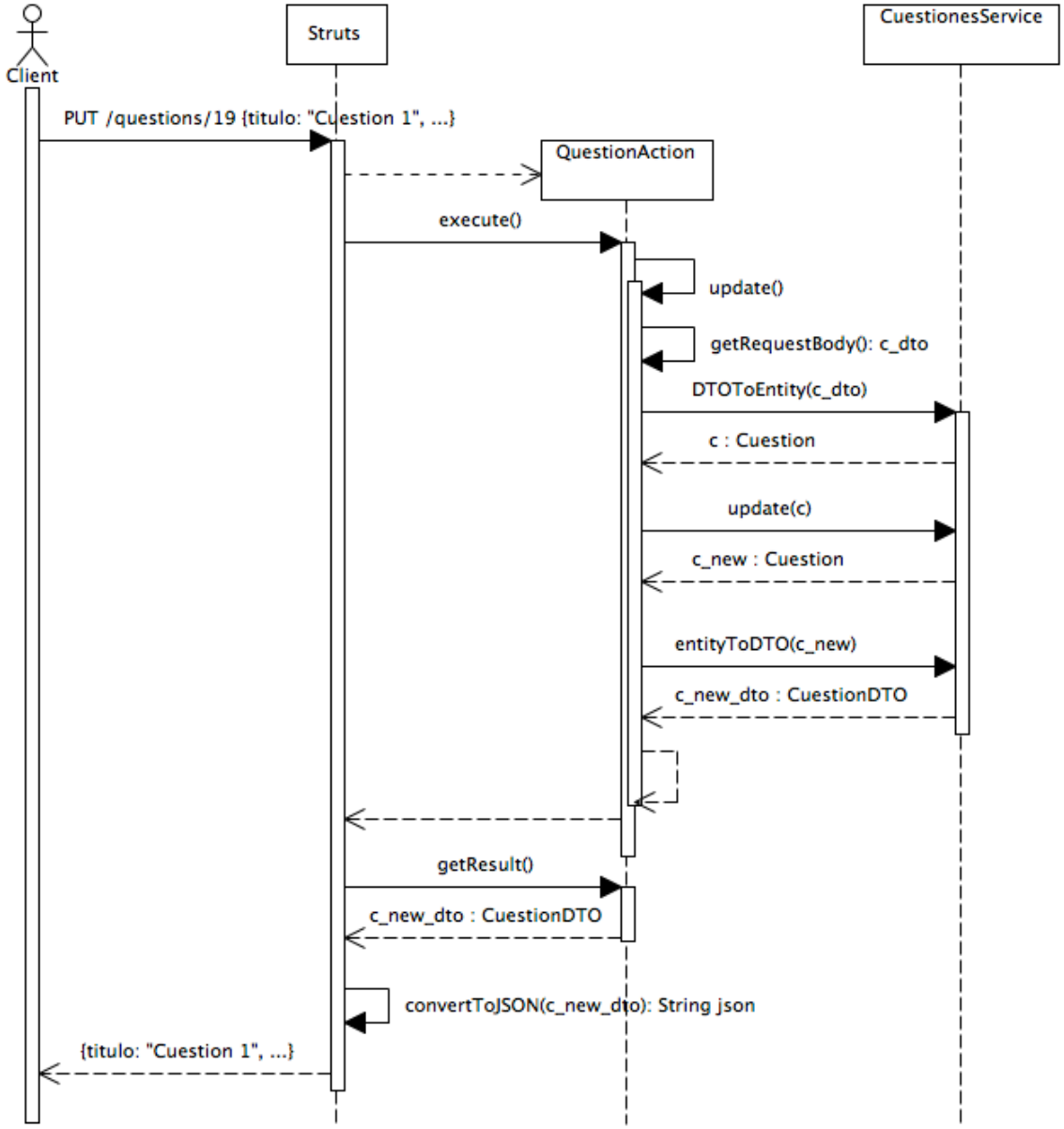


Figura 5.10 Petició de modificació d'una qüestió. Des-serialització de la petició, resolució del problema, i serialització de la resposta.

5.6 Autenticació, connexió a la base de dades i accés a la capa de domini (*auth*)

Un dels requeriments del projecte és que l'autenticació de l'usuari s'ha de fer contra la els usuaris de l'*SGBD*. Si no hi ha usuari autenticat, no hi ha connexió a la base de dades.

Això implica que, perquè l'aplicació pugui connectar-se a la base de dades, per una banda necessitem que l'usuari ens pugui donar les seves credencials, i per l'altra necessitem tenir emmagatzemades la resta de dades de connexió en algun lloc del projecte.

- Les dades de connexió es troben al fitxer `conexion.properties` a l'arrel del projecte.
- Les credencials de l'usuari les recollim a través de la pantalla d'accés, que és responsabilitat de l'acció `auth.LoginAction`. Aquesta acció recull les credencials de l'usuari i intenta connectar-se a la base de dades amb les dades de connexió configurades al fitxer `conexion.properties`. Si la connexió és exitosa, les credencials de l'usuari es desen en sessió per a que les properes peticions es puguin connectar a la base de dades sense requerir interacció altre cop.

Les *Action*s amb accés restringit de l'aplicació, doncs, són les que necessiten estar connectades a la base de dades per dur a terme la seva funció, i, per tant, necessiten que l'usuari s'hagi autenticat primer.

El mecanisme que s'usa, entre d'altres coses, per restringir l'accés a aquestes accions és un *Interceptor* d'*Struts* que hem creat per a l'ocasió, el `auth.DbConnectionInterceptor`, i que hem col·locat a la pila de pre-procés dels *packages* d'accions `secured` i `secured-ws`. Aquest interceptor s'encarrega de:

1. Mirar si hi ha les credencials de l'usuari a la sessió.
    a. Redirigir a la pantalla d'accés si no n'hi ha.
2. Connectar amb la base de dades.
3. Inicialitzar la capa de dades passant-li la connexió.
4. Inicialitzar la capa de domini passant-li la capa de dades.
5. Injectar la capa de domini a l'*Action* per a què aquesta la pugui fer servir per resoldre la petició.

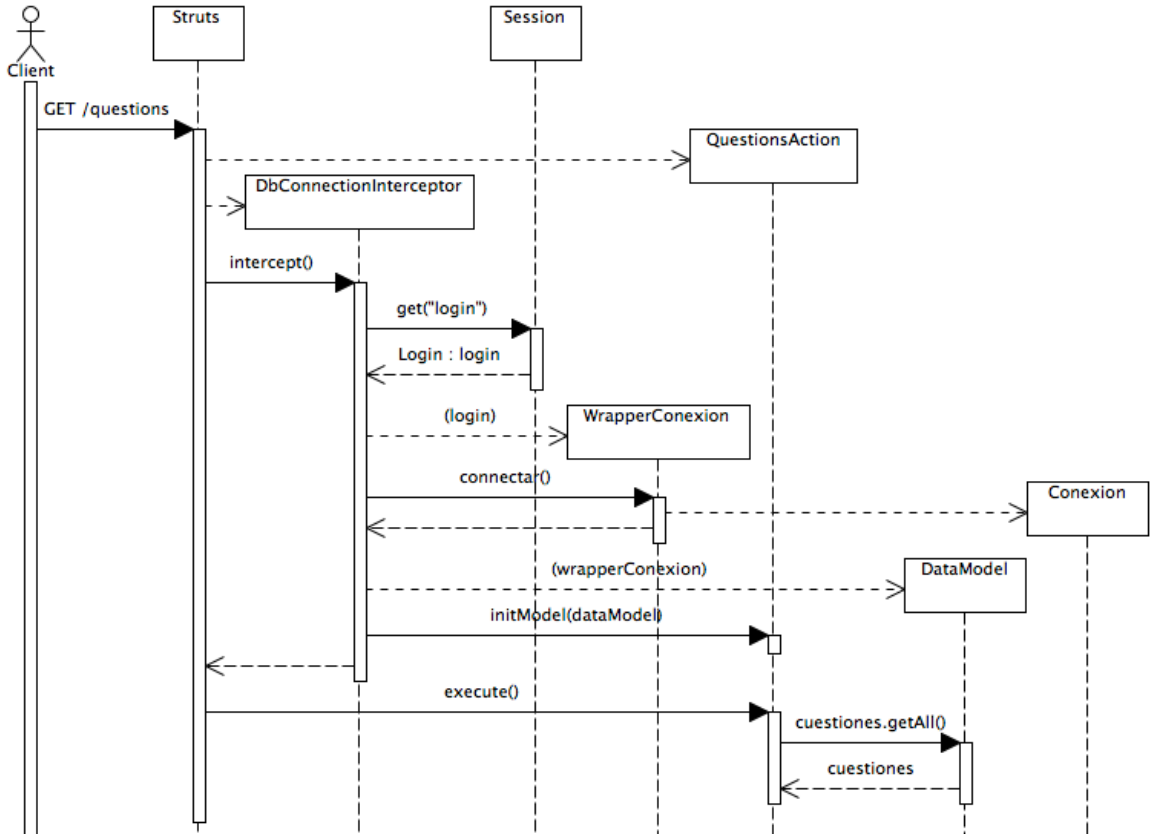


Figura 5.11 Petició qualsevol d'un usuari autenticat al sistema. Conexió a la base de dades i creació del Model.

Per fer disponible tota la capa de domini a les *Action*s, es crea l'objecte `modelo.DataModel`. Aquest conté tots els *Service*s per poder executar qualsevol operació del domini.

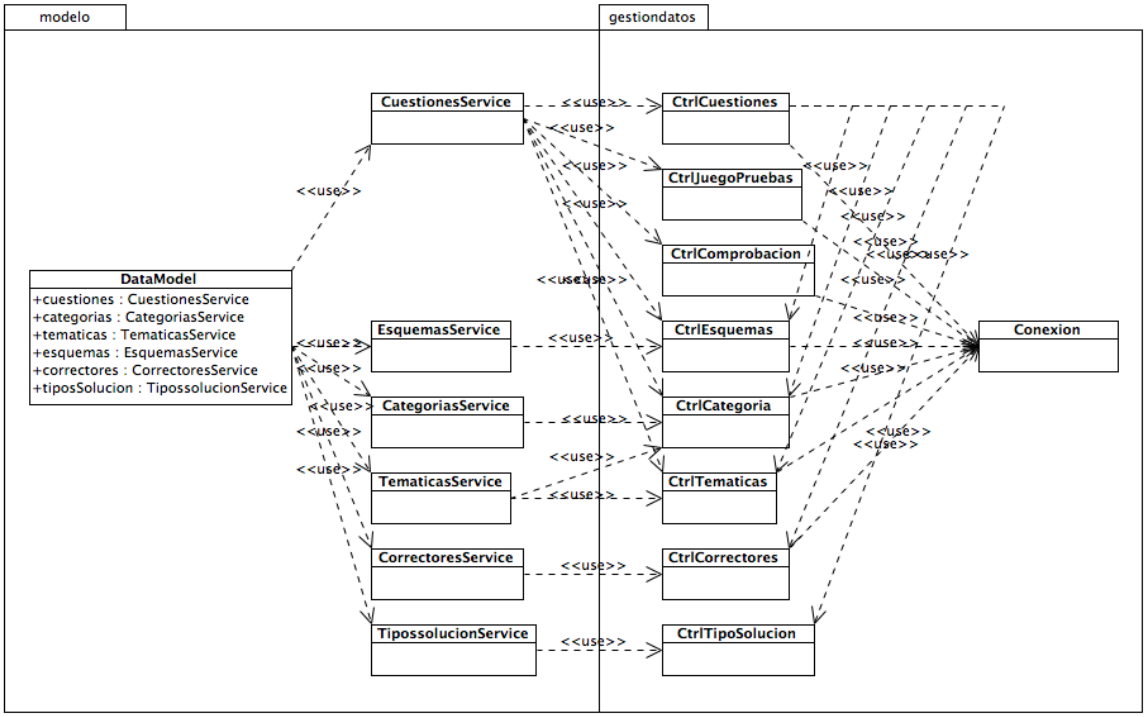


Figura 5.12 Mostra l'objecte `DataModel` *com a agregador de la capa de domini, i les seves dependències, fins a arribar a la connexió a la base de dades.*

Capítol 6: Disseny i implementació: Client

6.1 Arquitectura

És pot dir que l'arquitectura necessària en l'aplicació client és molt similar a la que seguia l'aplicació anterior, amb la diferència que la capa de persistència de dades ara ha de comunicar-se amb els serveis webs de l'aplicació servidor, enlloc d'una base de dades.

Així doncs, l'aplicació client s'organitza conceptualment de la següent manera:

- **Capa de presentació**: es responsabilitza de la interacció amb l'usuari, tant per recollir les seves accions com per mostrar l'estat de l'aplicació.
- **Capa de domini**: conté la lògica de negoci i manté l'estat de l'aplicació.
- **Capa de gestió de dades**: és l'encarregada de conèixer l'origen de les dades de l'aplicació i com obtenir-les i modificar-les. En aquest cas, aquest origen de dades és l'aplicació servidor.

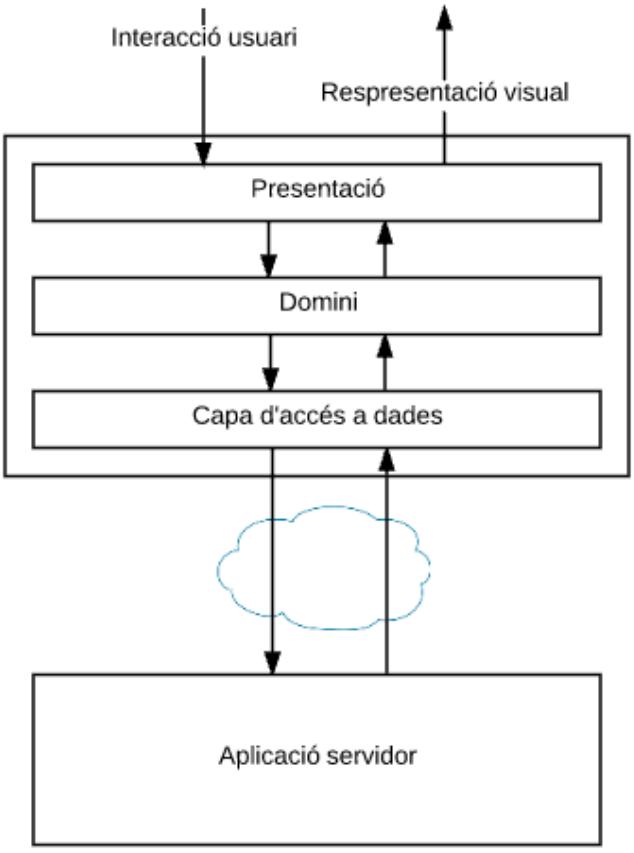


Figura 6.1

6.2 Mòduls

La quantitat de "components" que hi ha a l'aplicació, especialment a la capa de presentació, és bastant elevada, i fa difícil l'organització i representació del sistema. Per a solucionar aquest inconvenient, *AngularJS* ens proporciona la possibilitat d'organitzar, a més a més, l'aplicació en Mòduls. Els mòduls són una agrupació transversal de components amb una temàtica en comú, sense importar la seva responsabilitat en el sistema. Així doncs, en un mateix mòdul hi podem tenir components de la capa de presentació i de la capa de domini tots junts.

L'aplicació s'ha organitzat en els següents mòduls:

- `App`: agrupa els components que són essencials per l'arrancada i la configuració de l'aplicació.
- `Commons`: agrupa tots els components que són útils per a la resta de mòduls.
- `Questions`: agrupa tots els components relacionats amb els casos d'ús de *Qüestió*.
- `Categories`: agrupa tots els components relacionats amb els casos d'ús de *Categoria*.
- `Thematics`; agrupa tots els components relacionats amb els casos d'ús de *Temàtica*.
- `Schemas`: agrupa tots els components relacionats amb els casos d'ús d'*Esquema*.

6.3 Accés a dades

Les dades de negoci del sistema estan centralitzades al servidor. L'aplicació servidor exposa aquestes dades posant a disposició un ventall de peticions disponibles sobre el protocol *HTTP(s)* per a què aplicacions clients les puguin consumir.

Això ens permet, des de l'aplicació client, obtenir totes les qüestions, categories, temàtiques i esquemes del sistema, i comunicar-ne de noves, o comunicar canvis sobre aquestes que s'hagin de persistir.

Es poden veure les especificacions de les peticions disponibles a la secció 6.1 d'aquest document.

Per a facilitar el procés, asíncron, de comunicació amb el servidor, usarem la llibreria *Restangular*[16], que ens ofereix una capa d'abstracció sobre les comunicacions *HTTP(s)*, una bona gestió dels errors, i integració amb el cicle *MVC* de detecció de canvis d'*AngularJS*.

16

6.4 Domini

La capa de domini de l'aplicació client s'ha d'ocupar de:

1. Mantenir l'estat de les dades de negoci de l'aplicació.
2. Mantenir l'estat global de certs components específics de la interfície d'usuari.

6.4.1 Mantenir l'estat de les dades de negoci de l'aplicació (Diccionaris i Services)

D'aquest objectiu se n'encarreguen els Diccionaris. Els diccionaris fan vàries funcions:

1. Obtenir les col·leccions de les entitats de negoci usant la capa d'accés a dades per a comunicar-se amb l'aplicació servidor.
2. Mantenir aquestes col·leccions i oferir-les a la resta de l'aplicació.
3. Fer un seguiment de les modificacions que s'hi fan a aquestes col·leccions (nous elements, canvis, eliminacions).
4. Permetre tirar enrere les modificacions que s'hi han a aquestes col·leccions.
5. Persistir aquestes col·leccions d'entitats de negoci usant la capa d'accés a dades per a comunicar-se amb l'aplicació servidor.
6. Agrupar funcionalitats relacionades amb les entitats que formen les col·leccions.

Existeix un diccionari per a cada entitat de negoci del sistema. No obstant, per proporcionar a tots els diccionaris tota aquesta funcionalitat comuna, s'usa la classe `BaseDic` que es pot trobar al mòdul `Commons` de l'aplicació, de la qual extenen.

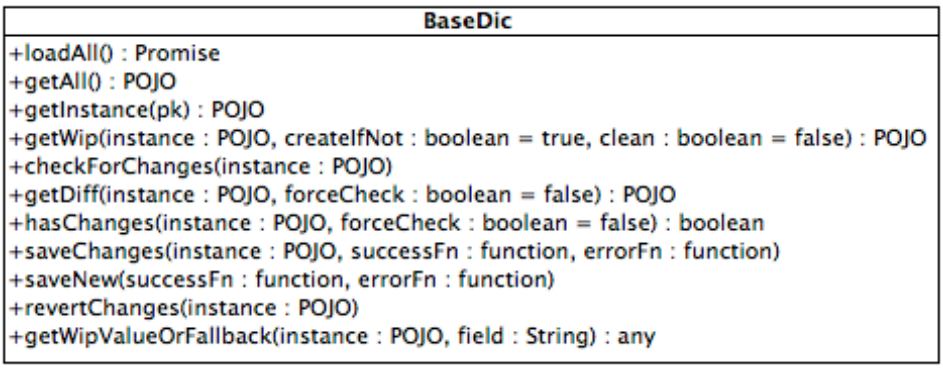


Figura 6.2 Especificació del diccionari base.

Si parem atenció, els diccionaris treballen amb *POJO*s (*Plain Old Javascript Objects*), o, el que és el mateix, objectes simples de *Javascript*. *AngularJS* treballa molt bé amb

objectes simples, així que no s'ha vist la necessitat de crear classes específiques per a modelar cada una de les entitats del sistema. En altres paraules, es treballa amb el *DTO* que ve directament del servidor. D'altra banda, les operacions que es poden executar sobre una entitat són al seu diccionari corresponent.

En el cas de només necessitar agrupar funcionalitats relacionades amb alguna entitat, sense totes les funcionalitats que el `BaseDic` ofereix, creem Services enlloc de diccionaris. Aquest és el cas dels jocs de proves i les comprovacions d'una qüestió, i amb tal finalitat existeixen el `TestsetsService` i el `ChecksService`.

Veiem a continuació un exemple simple de com s'utilitza un Diccionari per a una entitat del sistema:

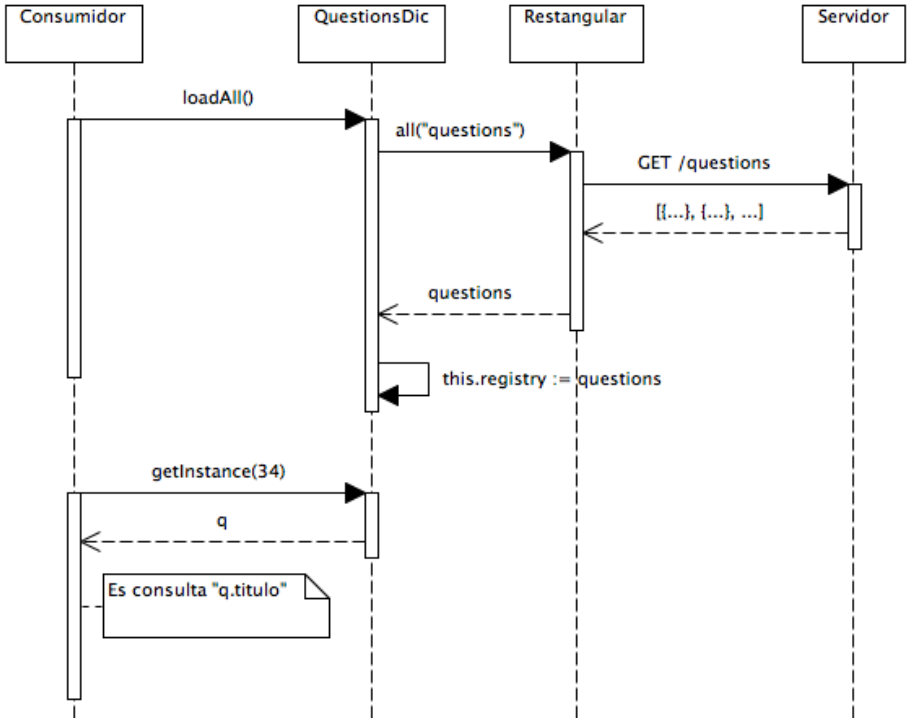


Figura 6.3

Com podem veure, el diccionari ha d'haver estat omplert amb les dades provinents del servidor abans de poder treballar amb ell. Més endavant veurem com s'omplen tots els diccionaris abans d'iniciar l'aplicació.

D'altra banda, la metodologia en la que s'està consultant l'entitat en el diagrama de seqüència anterior només s'ha de seguir en el cas que no tinguem intenció de modificar les dades obtingudes, o, vulguem consultar-ne les dades originals. Si el que es vol és obtenir una entitat on poder-hi fer canvis, s'ha de fer ús de les funcionalitats *WIP*.

6.4.1.1 *Work In Progress* (*WIP*)

A l'utilitzar les funcionalitats *WIP*, el diccionari crea una còpia de l'entitat per a poder treballar sobre seu sense preocupacions, sempre conservant la versió original obtinguda del servidor, per a poder tornar-hi sempre que es vulgui.

Vegem doncs, com s'han d'usar les funcionalitats *WIP* dels diccionaris per obtenir una còpia de l'entitat original i poder treballar amb ella, fins que decidim confirmar-la o tirar-la enrere:

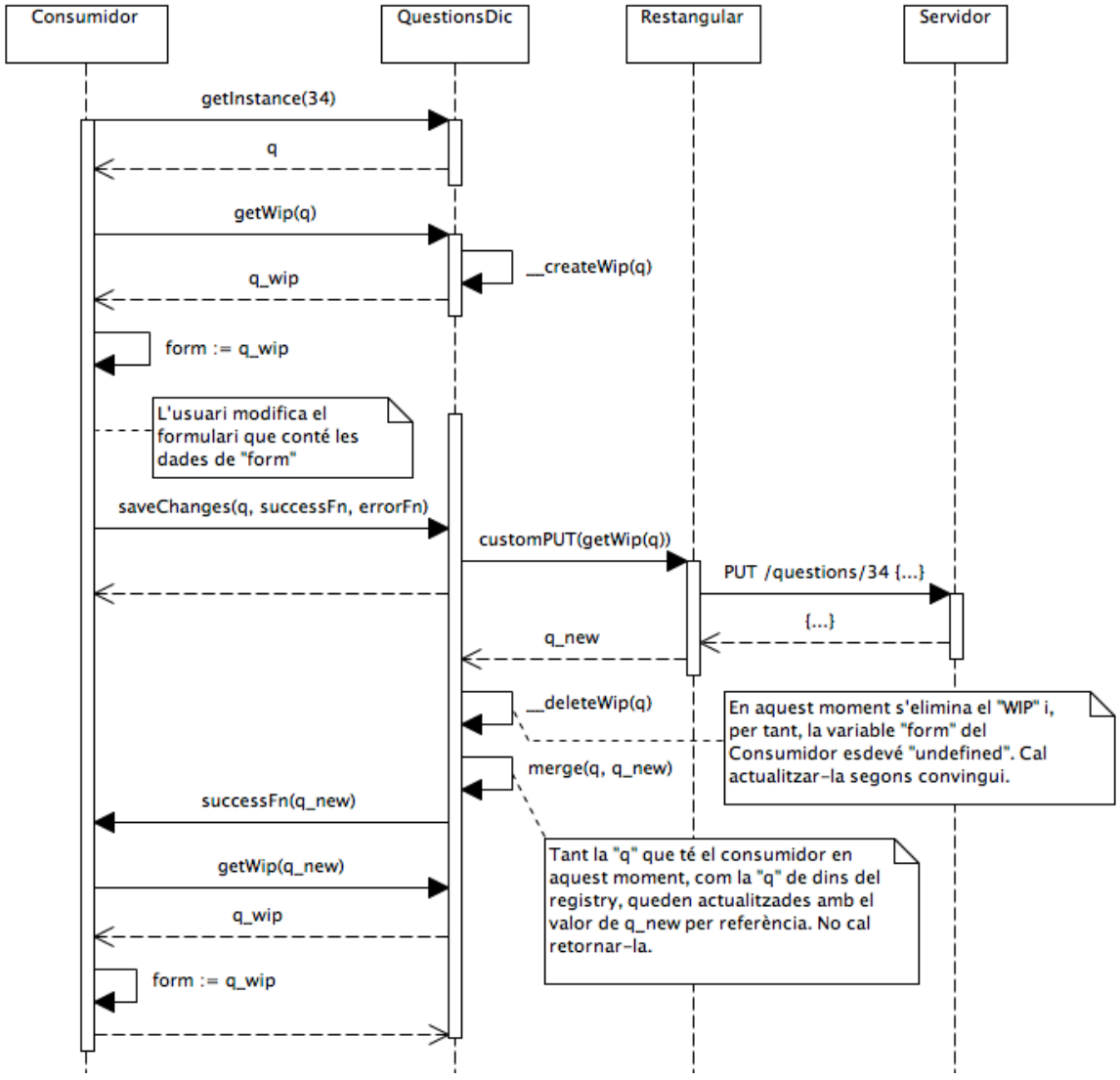


Figura 6.4 Cas d'ús d'obtenir una qüestió per a poder modificar-la, i confirmar-ne els canvis finalment.

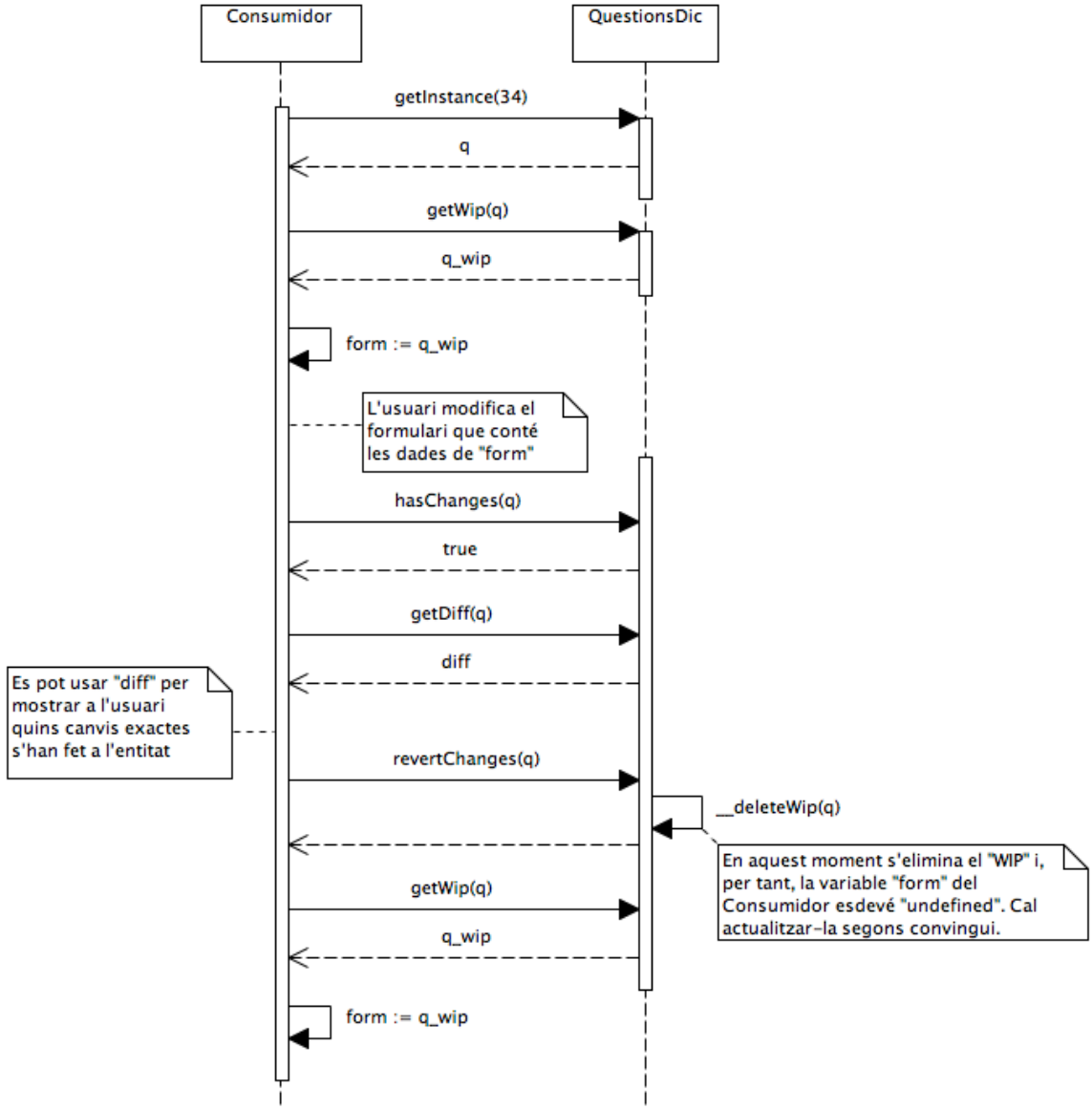


Figura 6.5 Cas d'ús d'obtenir una qüestió per a poder modificar-la, i tirar-ne enrere els canvis finalment.

6.4.2 Mantenir l'estat global de certs components específics de la interfície d'usuari.

Quan naveguem per les varies seccions de l'aplicació, les vistes que no hi ha necessitat de mostrar no s'amaguen, són destruïdes pel framework, estat de la vista inclòs. Això fa que haguem hagut de buscar una solució global i genèrica per a mantenir estats que després ens interessa recuperar un cop tornem a aquella secció (i la vista es construeixi altre cop).

Amb aquesta finalitat neix l'objecte `UiStateManager`, dins del mòdul `Commons`. Aquest objecte permet al desenvolupador desar informació i després recuperar-la, sense que sigui destruïda en cap moment durant l'ús de l'aplicació.

Un cas més específic és el del filtre de qüestions. En aquest cas, també es volen conservar durant l'ús de l'aplicació tots els paràmetres pels que l'usuari està filtrant el llistat de qüestions, per mantenir sempre aquest filtre fins que l'usuari el modifiqui. El cas d'ús de filtrat és bastant específic, ja que el filtre té uns valors per defecte, i s'ha de poder tornar a aquests valors per defecte sempre que es vulgui. L'objecte `UiStateManager` és massa genèric per a aquesta necessitat. Per tot això, també s'ha creat l'objecte `Search`, dins del mòdul `Questions`.

6.4.3 Visió general de la capa de domini

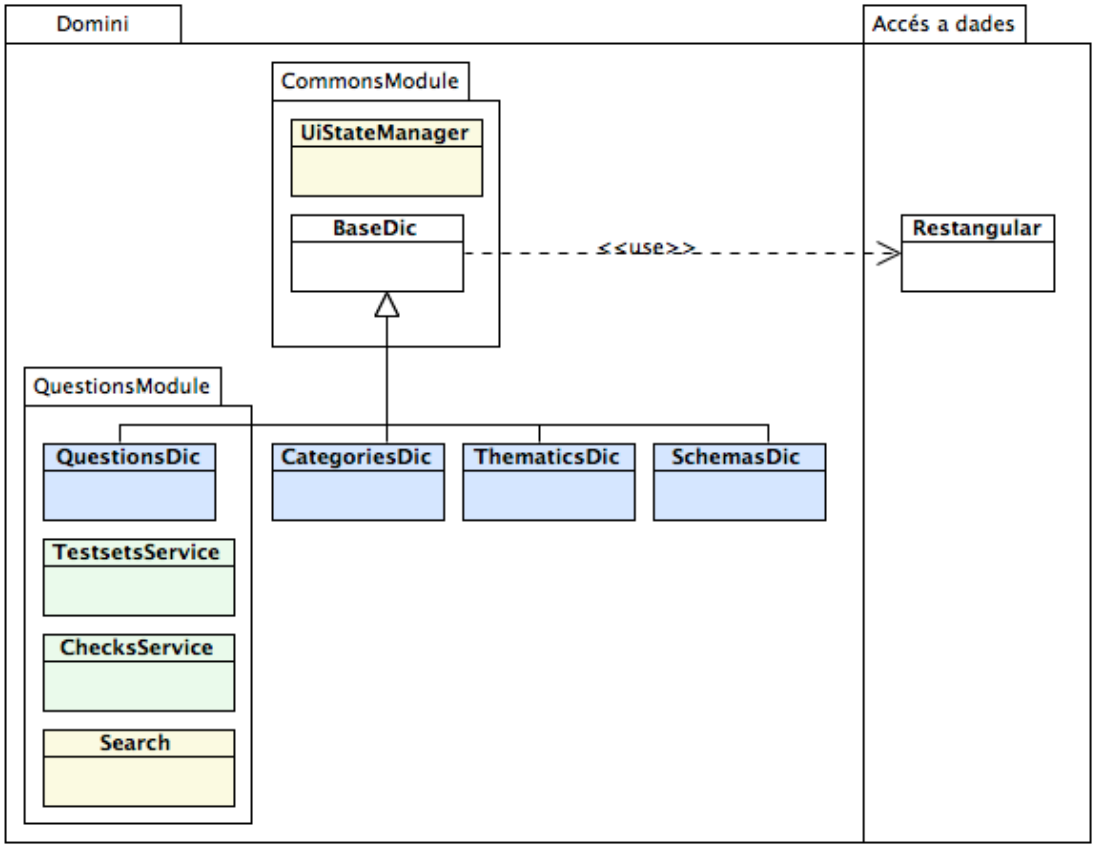


Figura 6.6 Visió general de la capa de domini

6.5 Capa de presentació

La capa de presentació és el punt d'entrada a l'aplicació client, mitjançant el qual l'usuari dona l'ordre d'executar les operacions que afecten el domini del sistema. També n'és el punt de sortida, mitjançant el qual l'usuari veu representat visualment el domini del sistema.

Tal i com s'ha vist al *Capítol 5*, la tecnologia escollida per a implementar l'aplicació client imposa el patró *Model-Vista-Controlador*, que afecta principalment a aquesta capa. Una bona estratègia per usar aquest patró és organitzar la capa de presentació en Components. Un Component equival a una unitat lògica, amb funcionalitat pròpia, que es mostra a l'usuari, i amb la qual aquest pot interactuar-hi. Cada component té una finalitat definida, i, alhora, pot contenir altres components. Els components estan formats per:

- una **Vista**: és un conjunt de sub-components posicionats en un cert ordre i disposició, que configuren la part visual de l'aplicació. Representa gràficament l'estat del Model i s'encarrega d'acceptar les accions de l'usuari i delegar-les al seu controlador específic. En el nostre cas, una vista és una plantilla *HTML*.
- un **Controlador**: és un objecte lògic que s'encarrega de donar sentit a les accions de l'usuari sobre la vista relacionada, i les tradueix en operacions cap al model.
- un **Model**: s'encarrega de mantenir l'estat de la vista.

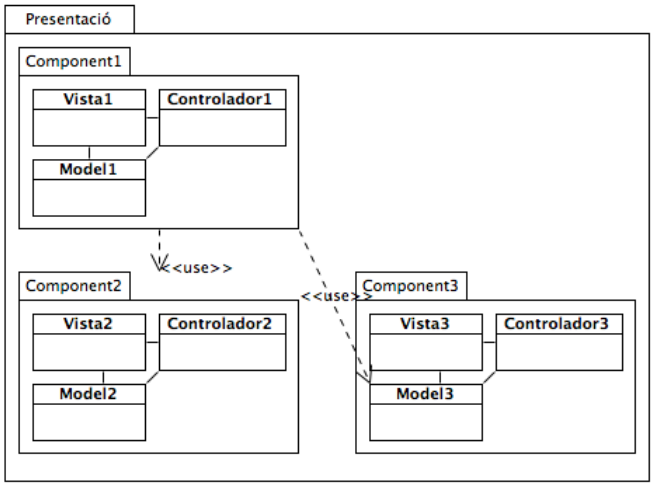


Figura 6.7 Esquema genèric de components MVC.

A continuació es mostra una successió d'esquemes que ajuden a entendre els Components de la capa de presentació que té l'aplicació, el seu propòsit, i les seves relacions entre ells.

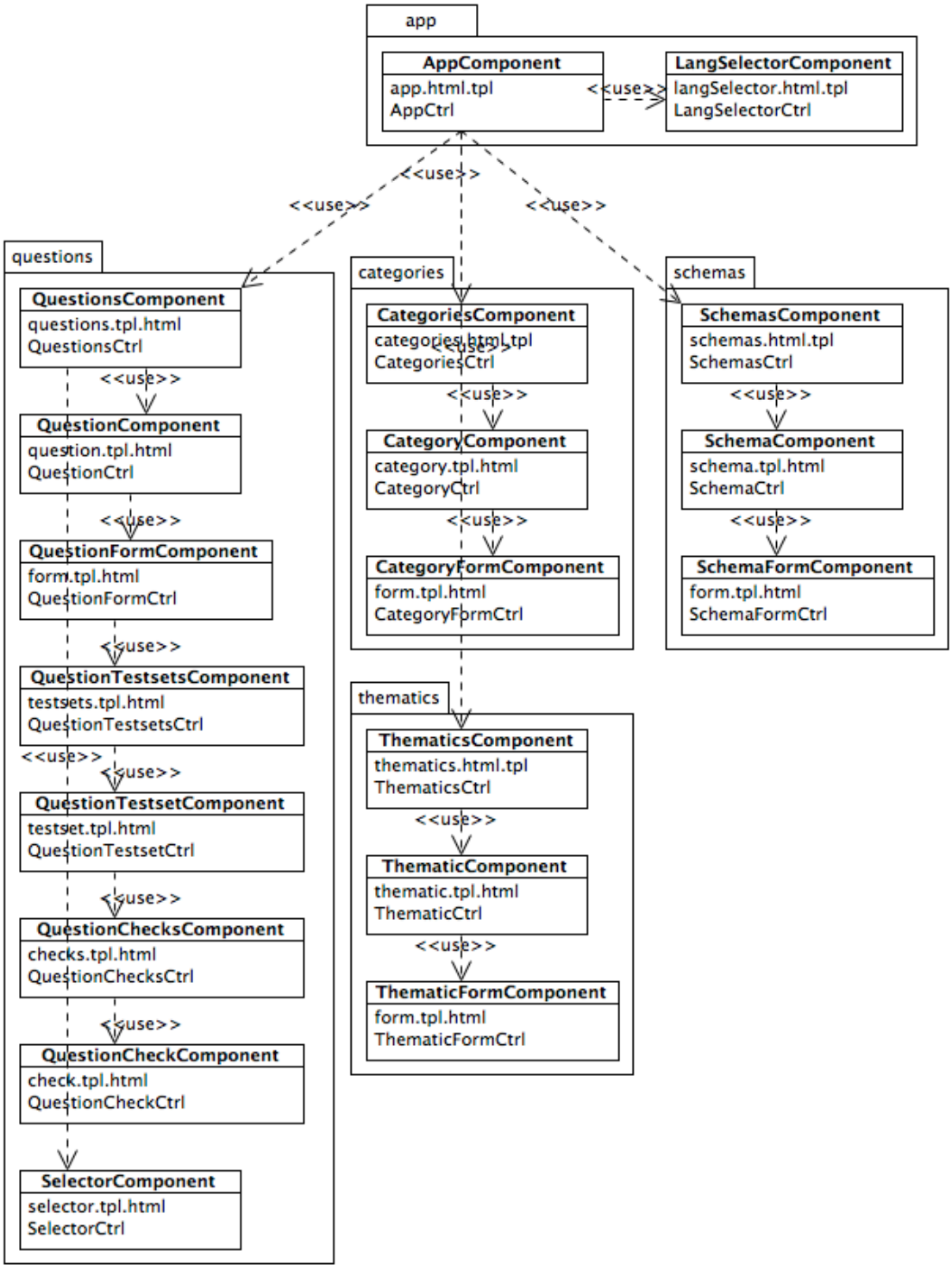


Figura 6.8 Components de la capa de presentació organitzats per mòdul, i les seves relacions d'ús.

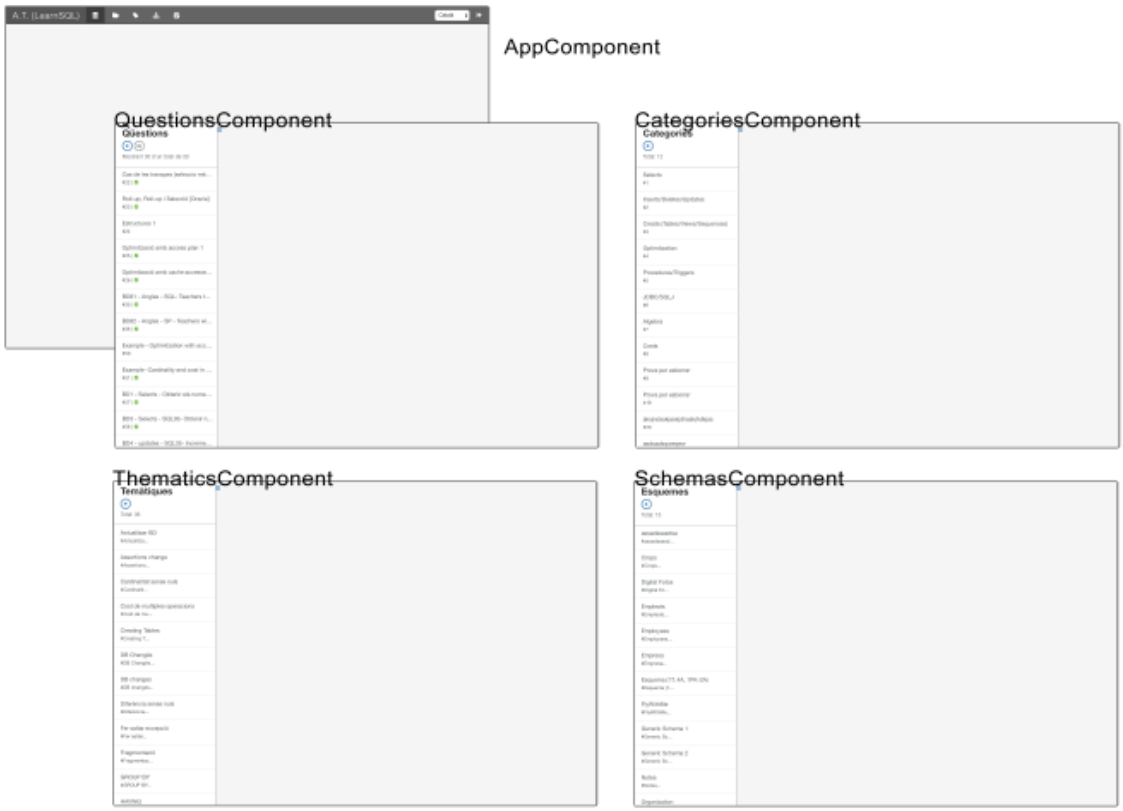


Figura 6.9 Aspecte visual de l'AppComponent, i dels 4 components directes que es poden mostrar dins seu segons la secció principal que l'usuari demani.

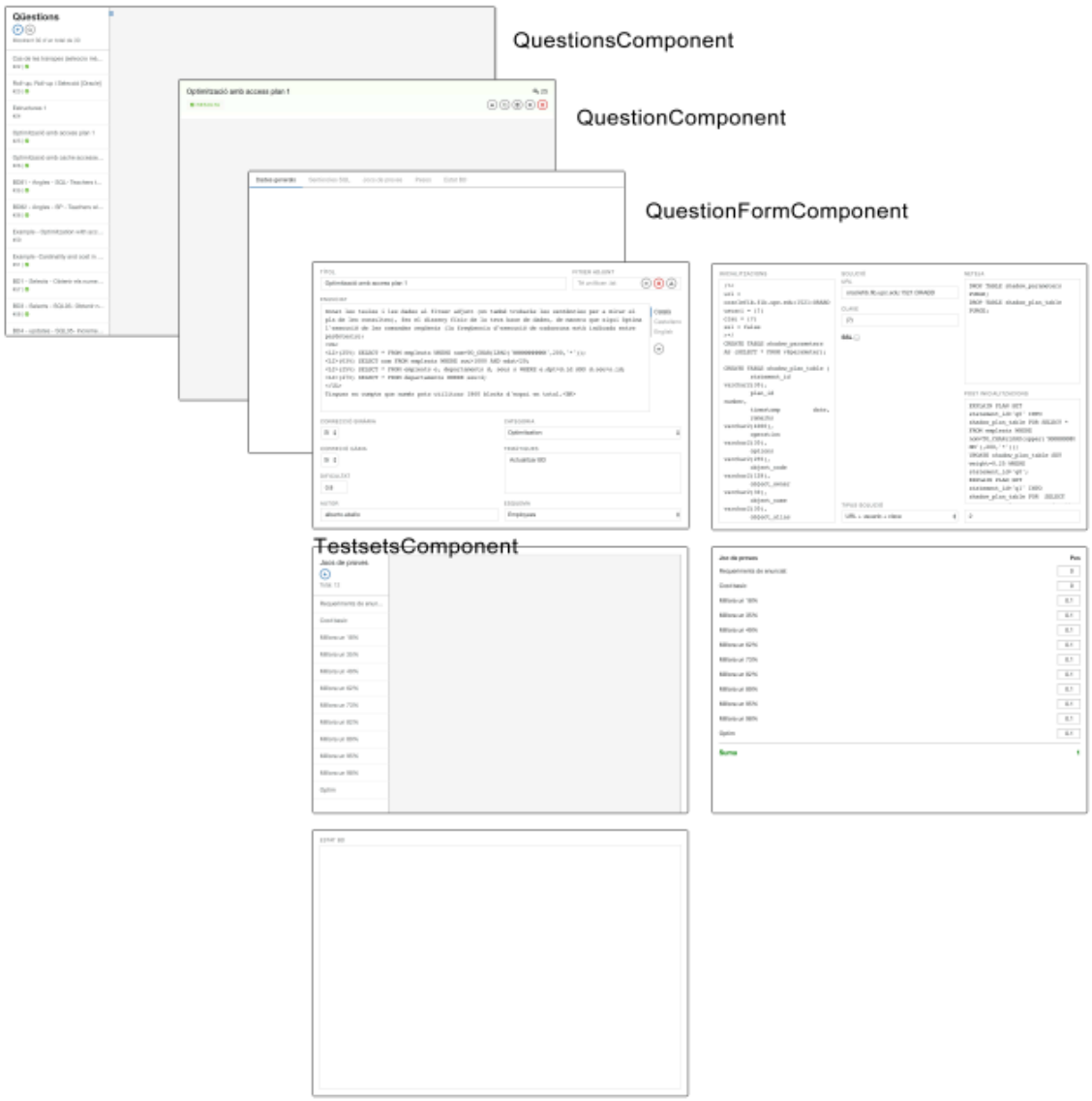


Figura 6.10 Arbre de components simplificat de la secció de gestió de qüestions

A continuació trobem una descripció de les responsabilitats dels Components més destacats de l'aplicació:

Component	**Responsabilitat**
`AppComponent`	Controla la distribució general de l'aplicació, el menú de seccions principals, i el selector d'idioma.
`Question`**`s`**`Component`	Controla el llistat de qüestions del domini, amb els seus indicadors d'estat, el filtre i la capacitat de seleccionar-ne una.
`QuestionComponent`	Controla la informació de la qüestió que està seleccionada, els seus indicadors d'estat, i els botons d'accions disponibles.
`QuestionFormComponent`	Controla el formulari de la qüestió, amb totes les seves diferents pestanyes, camps deshabilitats, camps dependents, etc. Controla també el procés de desar o cancel·lar canvis de la qüestió.
`QuestionTestset`**`s`**`Component`	Controla el llistat de jocs de proves de la qüestió seleccionada, la capacitat de seleccionar-ne un, i la d'afegir-ne.
`QuestionTestsetComponent`	Controla el formulari del joc de proves seleccionat, i la capacitat d'eliminar-lo.
`QuestionCheck`**`s`**`Component`	Controla el llistat de comprovacions del joc de proves seleccionat, la capacitat de seleccionar-ne una, i la d'afegir-ne.
`QuestionCheckComponent`	Controla el formulari de la comprovació seleccionada, i la capacitat d'eliminar-la.

Com s'aprecia a la *Figura 7.8*, cada component té la seva pròpia plantilla *HTML* (vista), i el seu propi controlador, que s'encarrega d'exposar un model de dades a la vista. Una vista només treballa amb el que el seu controlador li proporciona.

Podem veure, per exemple, què exposa el controlador del component `QuestionComponent` a la seva vista a continuació:

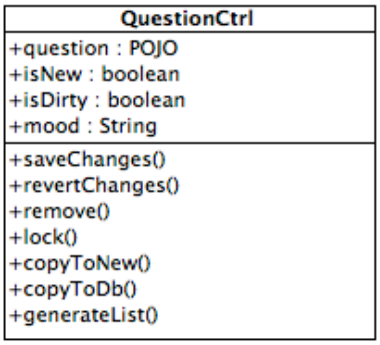


Figura 6.11 Propietats (model) i operacions que el controlador del QuestionComponent exposa a la seva vista

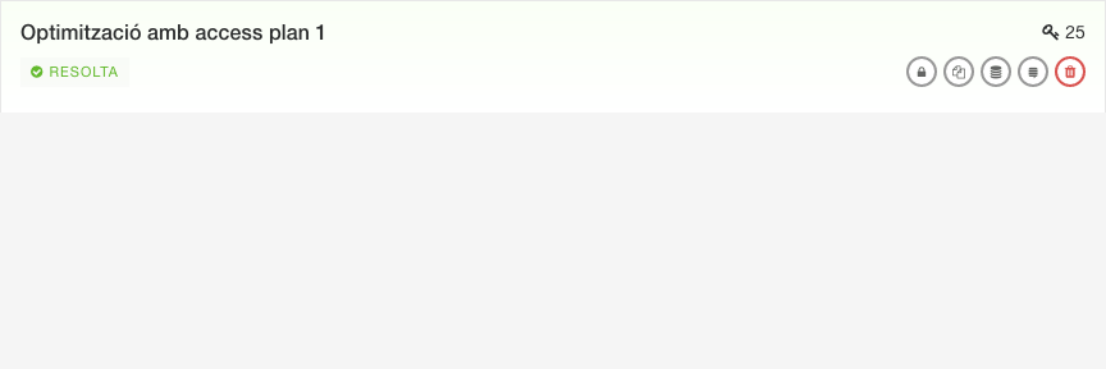


Figura 6.12 Vista del QuestionComponent

Es pot observar, doncs, que el controlador del component `QuestionComponent` exposa:

1. L'objecte sencer de la qüestió que està seleccionada en aquest moment, per a què la vista pugi mostrar informació com el títol o la clau primària.
2. Atributs derivats (per eficiència) de la qüestió seleccionada, que ajuden a la vista a saber quins indicadors de l'estat de la qüestió o quins botons d'accions ha de mostrar. (Aquest atributs derivats es calculen quan el component s'inicialitza, o quan la qüestió seleccionada canvia).
3. Operacions que la vista ha de cridar en cas que l'usuari premi algun dels botons d'accions relacionades amb la qüestió.

La implementació del patró *MVC* controlador ens assegura que quan alguna d'aquestes propietats exposades a la vista canviïn, la vista serà actualitzada degudament. No obstant, decidir quina qüestió està seleccionada no és responsabilitat del component `QuestionComponent`, sinó del component `Question`**`s`**`Component`, i, per tant la qüestió que exposa `QuestionComponent` mai canviarà. Per sort, els components poden comunicar-se mitjançant senyals. Si quan `Question`**`s`**`Component` canvia la qüestió seleccionada, s'envia una senyal a `QuestionComponent`, aquest serà capaç de re-calcular tot el que faci falta, i, si alguna cosa ha canviat, ara sí, la vista s'actualitzarà degudament. A continuació veiem tot aquest procés:

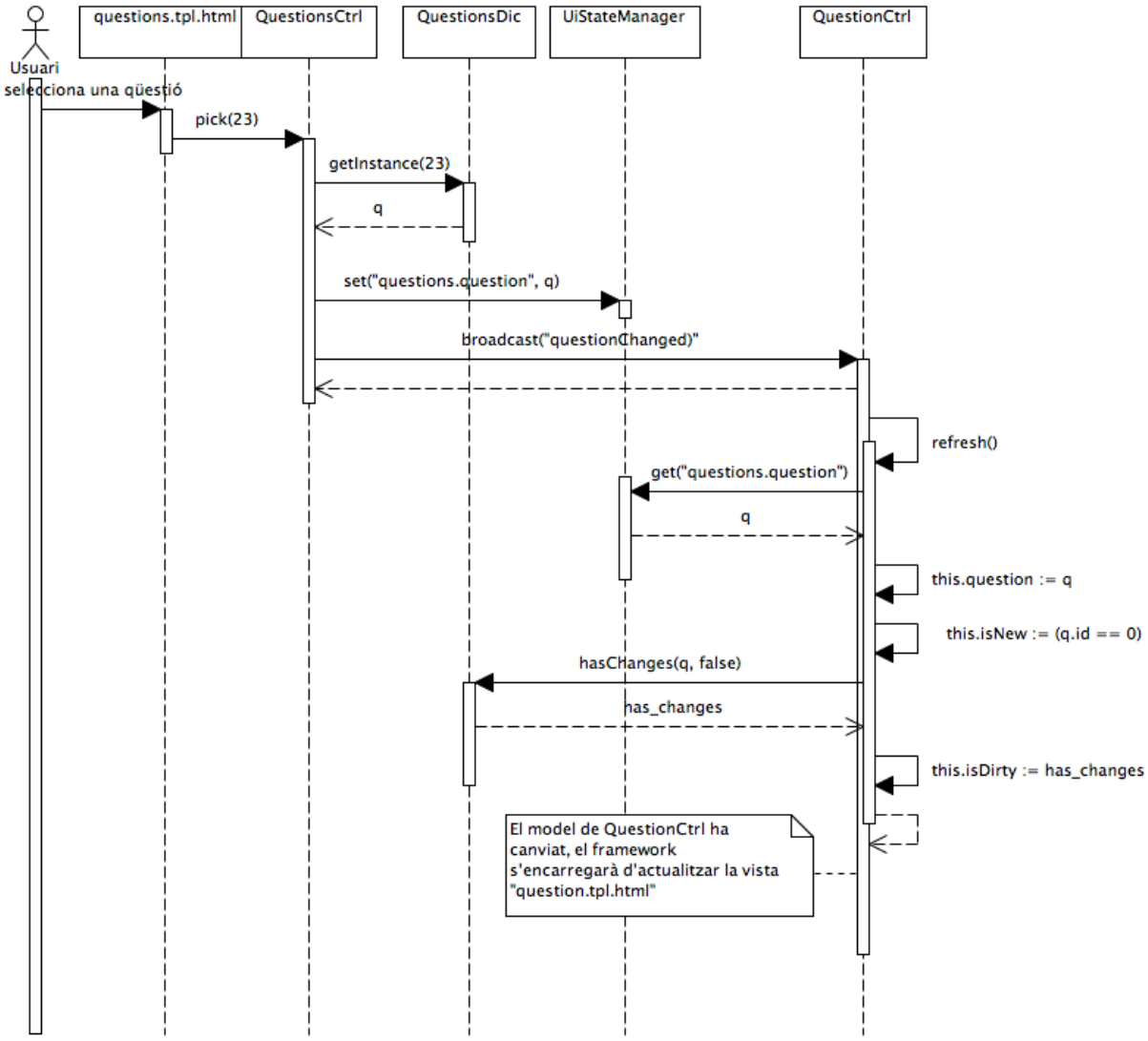


Figura 6.13

Es pot observar com es fa ús de l'objecte `UiStateManager` explicat anteriorment per desar l'entitat seleccionada i passar-la a altres components.

6.6 *Fetch* de les dades inicials

Quan l'aplicació client engega, una de les primeres coses que fa és demanar totes les dades de domini al servidor (qüestions, categories, temàtiques, esquemes i tipus de solució) i introduir-les en els diccionaris per a què els components no se n'hagin de preocupar més.

Aquest procés es du a terme quan l'aplicació intenta carregar l'estat bàsic, i es troba al fitxer `app.states.js` del mòdul `App`.

Un dels motius pels quals s'ha optat per aquesta política ha estat el de simplificar el codi dels controladors al màxim, sense haver de preocupar-se de peticions asíncrones. El projecte ja suposava un repte prou gran.

L'altre motiu és perquè l'experiència d'usuari millora molt d'aquesta manera. Per exemple, com que el filtratge sobre el llistat de qüestions es fa localment al client, sense haver de fer més consultes al servidor, l'experiència d'usuari és molt bona.

No obstant, el llistat de qüestions pot arribar a ser molt gran, ja que té molts elements dins (jocs de proves, comprovacions), i això es tradueix en molts MB de descarrega cada cop que s'inicia l'aplicació. Una possible millora pel projecte és la de continuar fent el fetch inicial de totes les dades, però amb només les dades que es necessiten pel llistat/filtrat. D'aquesta manera la mida de la descarrega es reduiria a gran escala. Quan l'usuari volgués consultar les dades d'una entitat, s'hauria d'anar al servidor a buscar-ne la resta.

6.7 Característiques destacades de la nova aplicació de client

Finalment, descrivim les característiques assolides amb la nova interfície d'usuari.

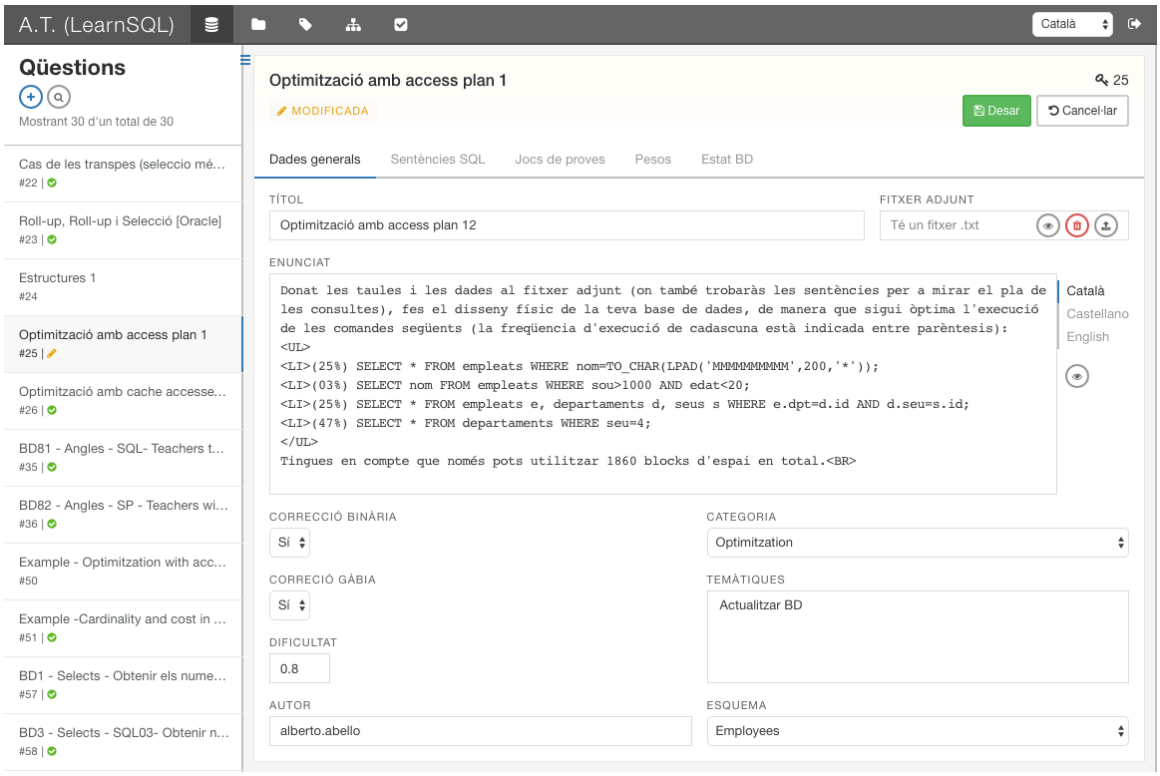


Figura 6.14 Distribució general de la nova aplicació, per referència.

1 | Es manté l'estat, en general, del que s'està fent.

L'aplicació segueix permetent els fluxos de treball en els que l'usuari deixa a mitges el que està fent, visita una altra entitat o secció, torna on era, i s'ho trobat tot tal i com ho havia deixat.

2 | La interfície de navegació cap a les seccions principals sempre està a l'abast.

La barra superior, que conté el menú de seccions principals, el selector d'idioma i el botó de sortir, sempre està a l'abast.

Figura 6.15

3 | Els llistats d'entitats sempre estan a l'abast mentre se'n consulta una, i és possible interactuar-hi.

Els hem col·locat sempre a la part esquerra de l'aplicació, tots amb el mateix format, per assegurar que el flux de treball és el mateix per a totes les entitats.

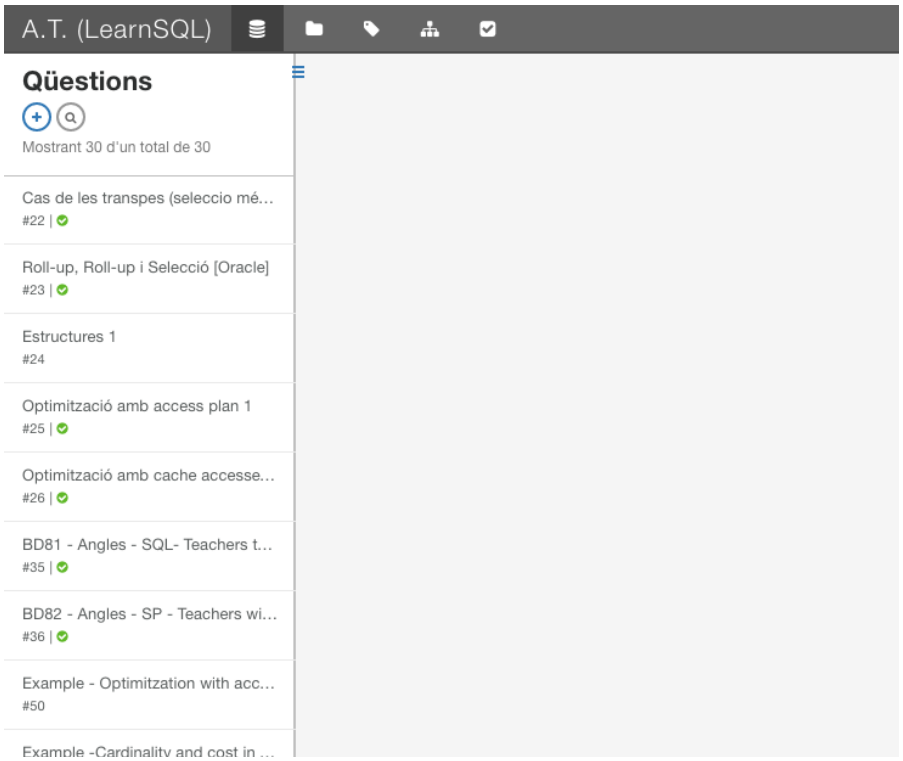


Figura 6.16

4 | Tots els elements importants relacionats amb el que s'està fent en un moment determinat estan visualment a l'abast.

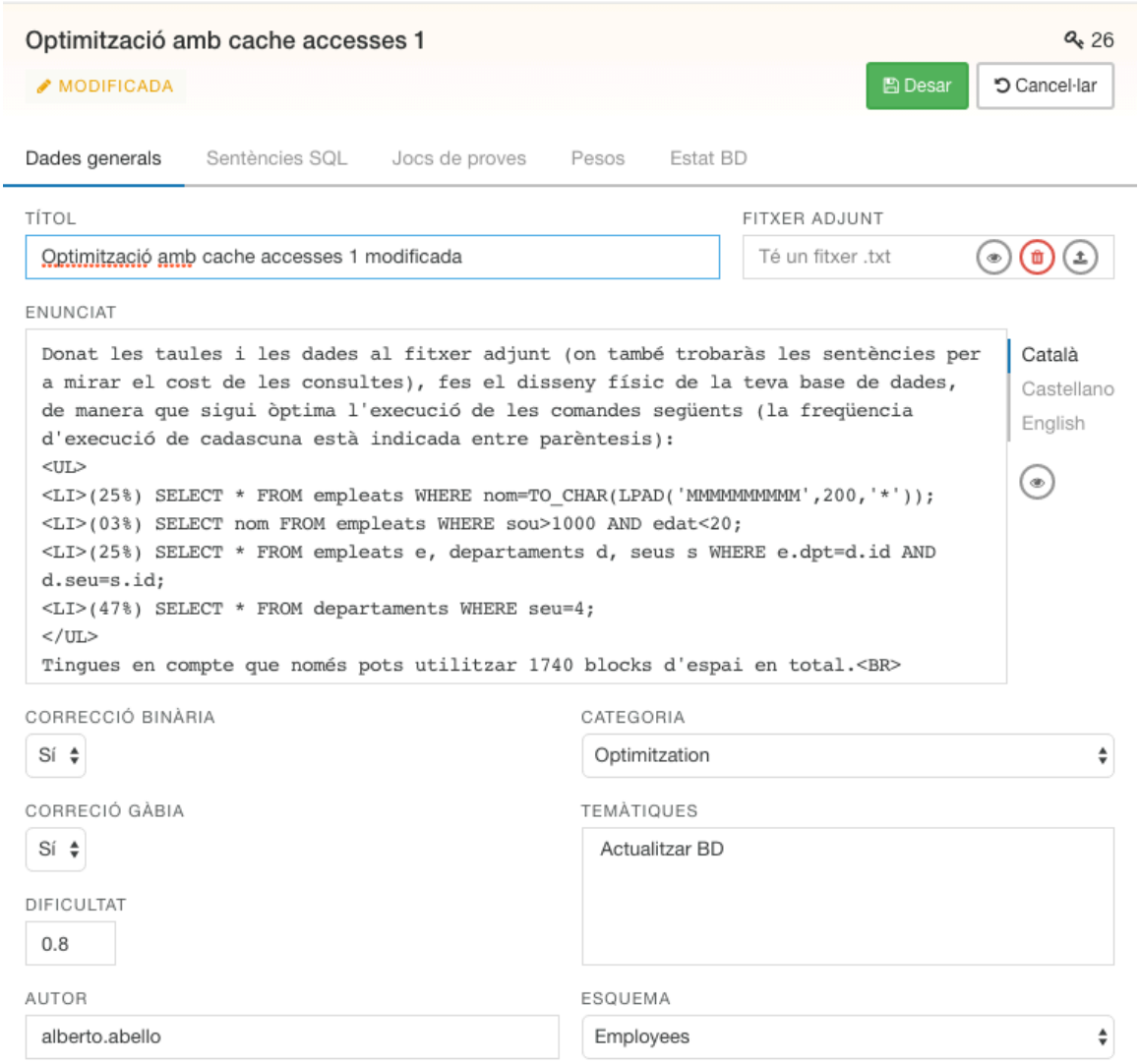


Figura 6.17 S'observen els botons d'accions, tots els camps de la secció, i la barra de títol, sempre visualment a l'abast.

5 | Nova capçalera d'entitat.

Combina les antigues barres de títol i d'accions, i les re-inventa fent-les molt més visuals i òptimes en espai.

Només es mostren les accions disponibles per l'estat actual, evitant distracció, i deixant visualment clar què s'està fent i què es pot fer.

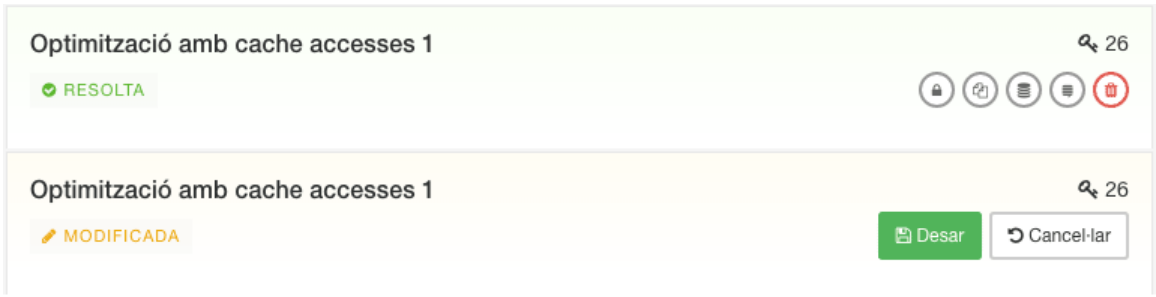


Figura 6.18

Els nous indicadors d'estat, cadascun amb el seu color i icona propis, consistents a través de tota l'aplicació, també contribueixen a la bona experiència.

Figura 6.19

Tant els botons d'accions com els indicadors d'estat reaccionen immediatament quan les accions de l'usuari ho requereixen.

6 | Nous indicadors d'estat també a les barres laterals de llistats d'entitats.

Ara és molt més fàcil veure quines qüestions estan resoltes, quines modificades, etc. només mirant la barra lateral de llistat d'entitats. I també reaccionen immediatament quan les accions de l'usuari ho requereixen.

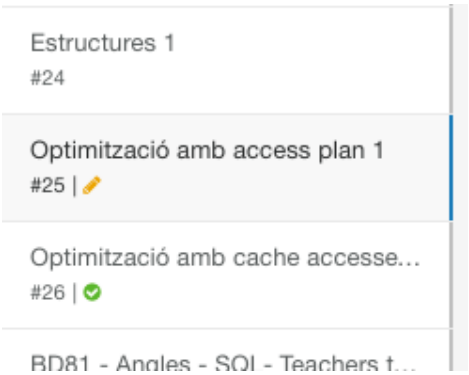


Figura 6.20

7 | Cancel·lació de canvis no confirmats.

L'aplicació segueix mantenint la versió confirmada de cada entitat, i permet tirar enrere els canvis que s'hi hagin fet d'una manera immediata.

8 | Seguiment dels canvis efectuats sobre una qüestió.

L'aplicació segueix anotant quins canvis específics s'han efectuat sobre una qüestió, i, al voler-la confirmar o restaurar, l'aplicació informa a l'usuari de quins han estat els canvis, donant a l'usuari la seguretat suficient per fer el pas final.

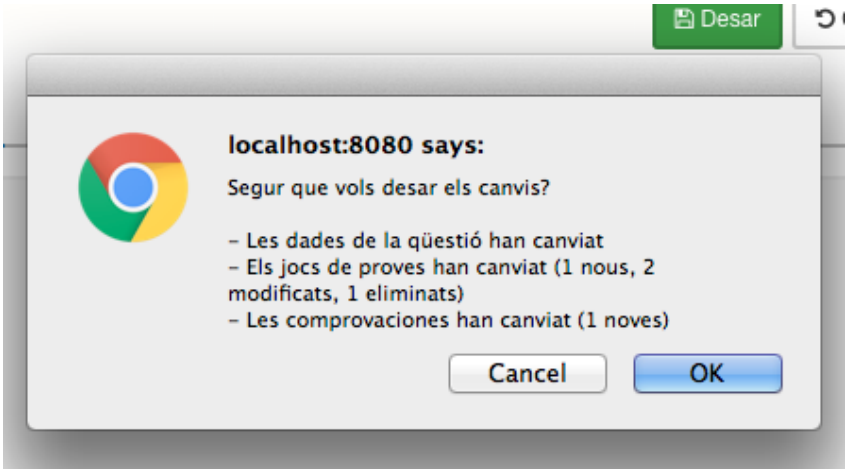


Figura 6.21

9 | Precisió en la comunicació d'errors de validació en una qüestió.

Quan hi ha un error de validació de dades al confirmar els canvis fets sobre una qüestió, l'aplicació segueix presentant a l'usuari la secció exacta on es troba aquell error.

10 | S’aprofita tot l’espai visual disponible.

L’aplicació segueix estant pensada per ocupar sempre tot l’espai del que disposa. Quan la finestra és més gran del compte, els elements que més espai necessiten s’expandeixen per aprofitar l’espai addicional.

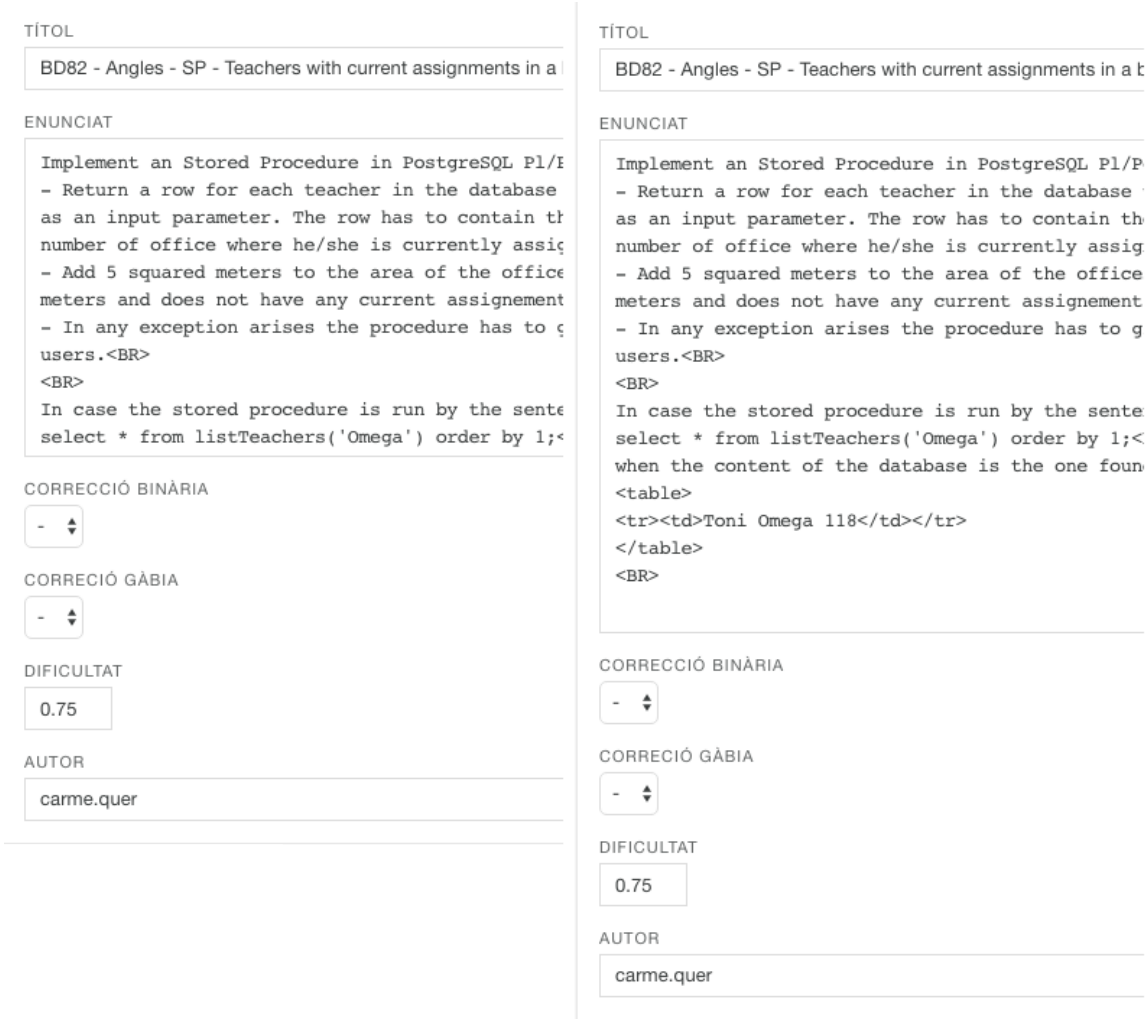


Figura 6.22 L’àrea de text de l’enunciat creix verticalment per donar visibilitat a més contingut.

11 | La barra lateral llistat d'entitats es pot amagar per a més espai de treball i concentració.

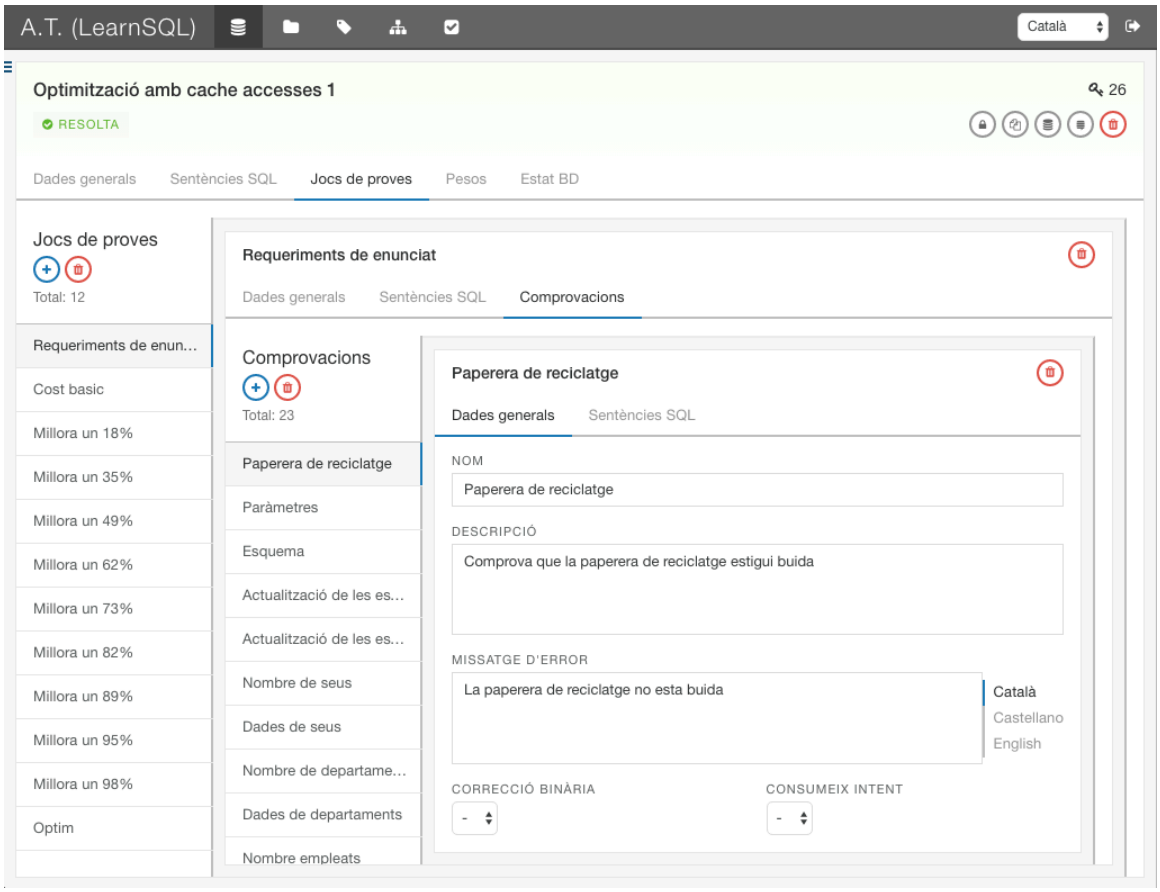


Figura 6.23

12 | La barra lateral llistat d'entitats es fa més ample en monitors més grans.

Si hi ha més espai a la pantalla, també pot haver-hi més espai per veure més informació sobre cada entitat al llistat.

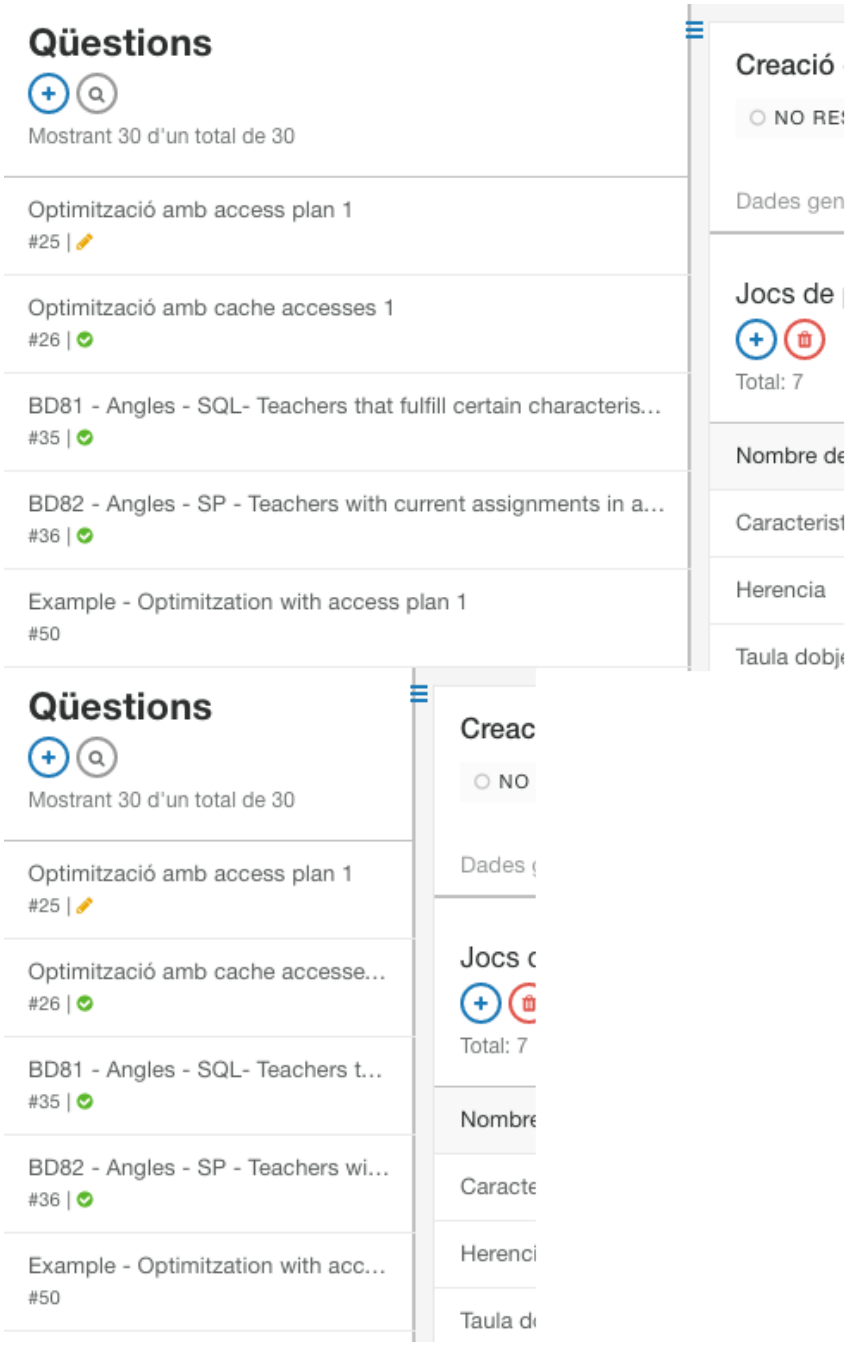


Figura 6.24

13 | Nou filtre de qüestions.

Ara, per defecte, el filtre de qüestions està amagat.

Al especificar-hi filtres, el llistat de qüestions s'actualitza instantàniament.

S'afegeix un nou tipus de filtratge per a poder veure només les qüestions que s'hagin modificat, però no confirmat.

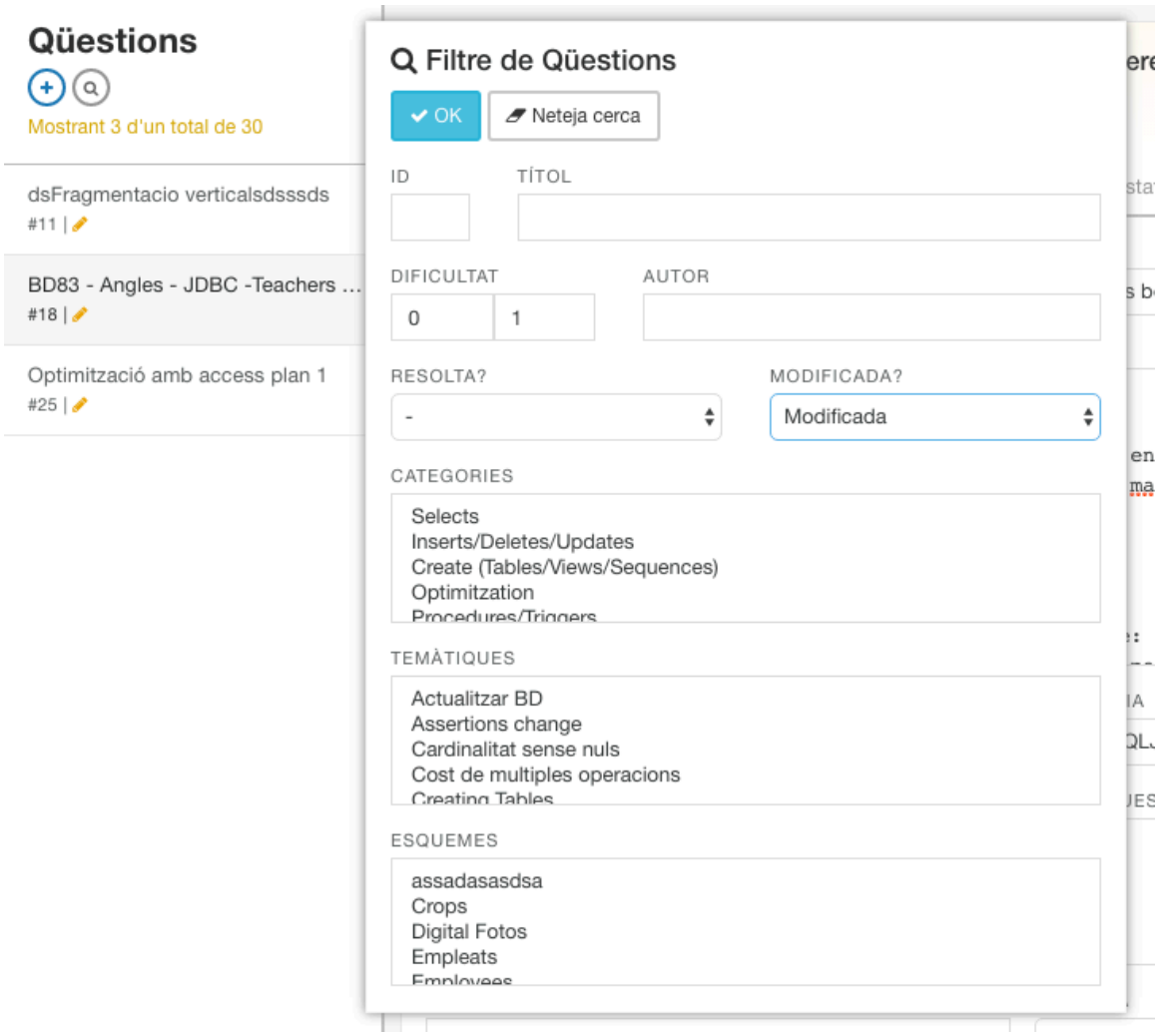
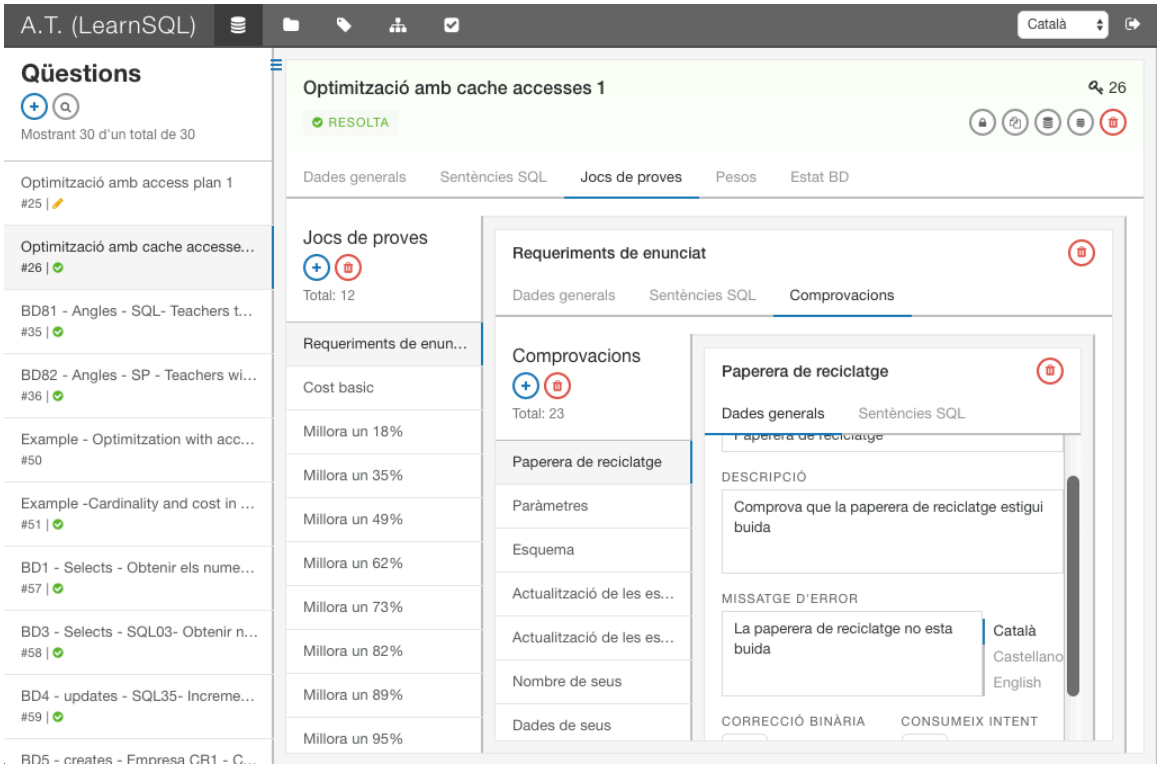


Figura 6.25

14 | L'aplicació està preparada per monitors petits.

L'aplicació està dissenyada per tenir tots els elements importants sempre visibles, però, quan la resolució de la pantalla és massa baixa, apareixen *scrolls* que permeten accedir a als mateixos contingut sense problemes. Es garanteix que cap element queda fora de la pantalla fins a una resolució mínima de 1024x600px.



15 | Ja es veuen a simple vista tots els jocs de proves que té una qüestió, i totes les comprovacions que té un joc de proves, quan s'estan consultant.

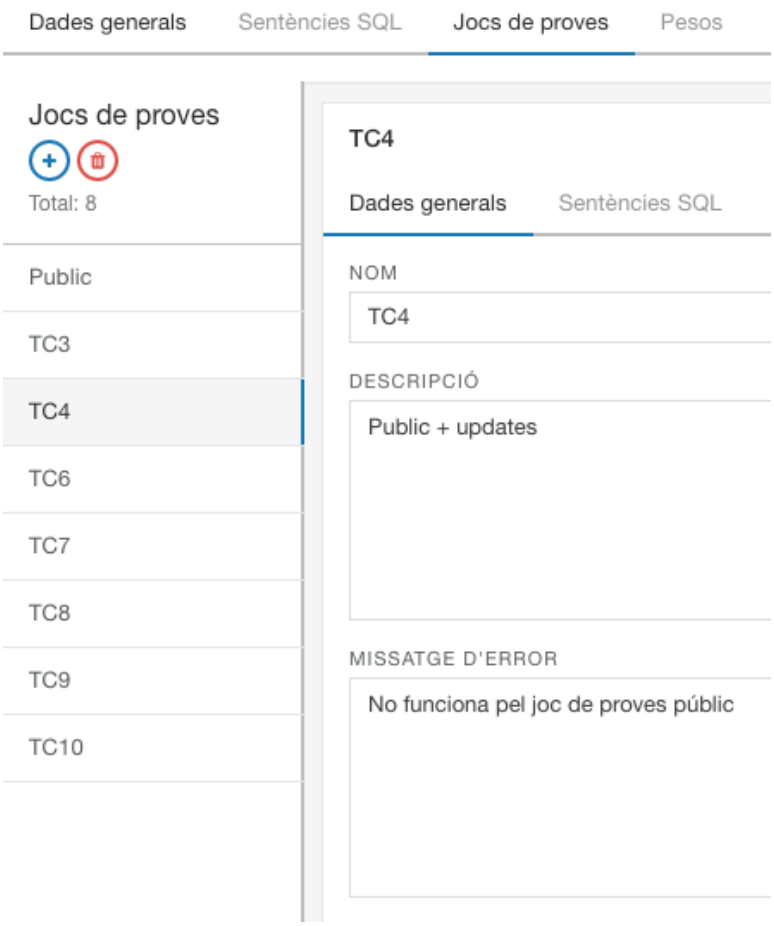


Figura 6.26

16 | Ja es veu a simple vista el sumatori dels pesos dels jocs de proves, i ja no impedeix desar la qüestió.

L'aplicació ja disposa d'una manera visual per veure quant sumen els pesos dels jocs de proves d'una qüestió. La suma dels pesos s'actualitza immediatament quan es canvia algun dels pesos.

A més a més, si no sumen 1 s'indica visualment, però es permet desar la qüestió.

Figura 6.27

Capítol 7. Estudi temporal i econòmic

7.1 Estudi temporal

Per a aquest projecte és impossible presentar una planificació inicial, ja que, per motius personals, s'ha anat desenvolupant al llarg d'un extens període de temps. No obstant, sí que s'ha fet una planificació de tasques a fer, i un seguiment de les hores invertides, que es presenta a continuació:

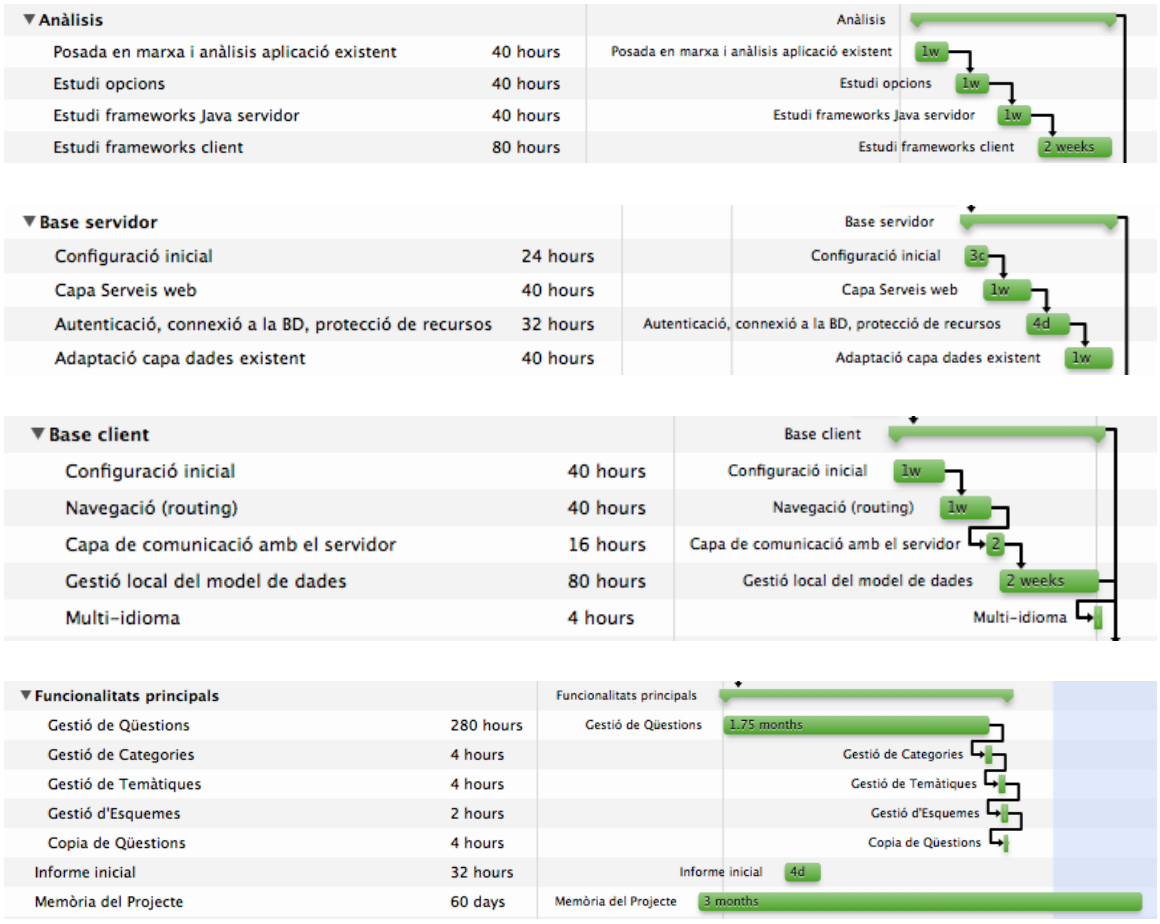


Figura 7.1

El projecte ha hagut de comptar amb una forta etapa d'anàlisi, al partir d'un projecte existent i voler-ne reproduir totes les funcionalitats en una altra plataforma diferent. També per estudiar les tecnologies disponibles per complir millor amb els objectius del projecte.

D'altra banda, l'etapa de desenvolupament de l'aplicació client ha ocupat la majoria del projecte, al ser la més complexa i la que requereix més detall, acumulant un total de 474 hores.

L'anàlisi, desenvolupament i la totals del projecte sumen un total de **970 hores**.

7.2 Estudi econòmic

Donat que totes les tecnologies que s'han usat al projecte són de llicència gratuïta o de codi lliure, només s'ha de tenir en compte el cost del *hardware* usat pel desenvolupament, i el cost del personal.

Per desenvolupar aquest projecte en un entorn professional s'haurien necessitat els següents perfils:

Perfil	**Cost/hora**	**Hores**	**Cost total**
Cap de projecte	60€	60	3.600€
Analista	45€	320	14.400€
UI/UX expert	30€	100	3.000€
Programador JAVA EE	30€	176	5.280€
Programador front-end	30€	314	9.420€
Totals		**970**	**35.700€**

Capítol 8. Conclusions finals

8.1 Objectius assolits

Els principals objectius del projecte s'han assolit de manera satisfactòria.

S'ha aconseguit portar tota la funcionalitat principal de gestió de l'*Authoring Tool* a la plataforma web, fet que fa que l'eina estigui disponible des de qualsevol ordinador amb connexió a internet i un navegador web. Així s'ha solucionat també els problemes de distribució de noves versions de l'aplicació, al ser la plataforma web l'encarregada de distribuir-les.

S'ha creat una interfície d'usuari clara i pensada per a facilitar la gestió complexa de les entitats del sistema, que preserva la bona experiència d'usuari que l'aplicació d'escriptori anterior tenia, i fins i tot la millora en certs aspectes. En el procés, s'ha donat especial importància a respectar els fluxos de treball que l'antiga aplicació permetia, reproduint-los en la mesura del possible a la nova aplicació web.

S'ha pogut re-usar la lògica de negoci de l'aplicació anterior, fet que assegura que el sistema segueix sent fiable i que no s'hi trobaran problemes anteriorment ja solucionats.

S'ha creat una arquitectura interessant, amb una justificada diversitat de llenguatges i metodologies, que busca l'especialització professional de cada una de les seves peces, amb pràctiques i patrons que fomenten la bona estructuració de l'aplicació i del seu codi font.

8.2 Feina restant i possibles millores

Tal com s'ha descrit al definir l'abast del projecte en el capítol introductori, queda pendent per a altres futurs projectes la implementació dels casos d'ús de gestió de correctors i execució.

En quant a possibles millores:

El *fetch* inicial de dades de l'aplicació client, o inclòs tota la política sencera, es podria optimitzar de la manera que es descriu a la secció 7.6 d'aquest document.

Per una altra banda, aquest *fetch* inicial de dades per part de l'aplicació client té la conseqüència de que un usuari pot estar potencialment treballant sobre dades que ja no són vàlides perquè algú altre les ha modificat amb una altra instància de l'aplicació. Els projectes anteriors no contemplaven tampoc una solució per a aquest problema, però aquest projecte proposa una idea:

Desenvolupar una comprovació que miri periòdicament (cada 2 minuts, per exemple), en segon pla, si ha canviat informació, com a mínim, de l'entitat que s'està consultant/modificant per l'usuari. Aquesta comparació es podria fer mitjançant *checksums*. Si l'entitat ha canviat, notificar a l'usuari de que no disposa de l'última versió de l'entitat, i proposar-li refrescar-la, o quedar-se, de moment, amb la que està veient.

Finalment, en algun moment es podria plantejar migrar l'aplicació client de *AngularJS 1* a *Angular2*[17]. Tot i que *AngularJS* 1 permet obtenir resultats molt bons amb una metodologia de treball molt agradable, hem vist que el rendiment es veu afectat a mesura que la quantitat de dades a gestionar creix. L'aplicació web que hem construït és totalment funcional fins i tot gestionant 1000 registres de qüestions alhora, però el rendiment es veu deteriorat a mesura que les col·leccions creixen molt. *Angular2* ha redissenyat tot el procés intern de detecció de canvis, per assegurar que es pot treballar amb quantitats grans de dades, però, malauradament, en el moment en que es va començar aquest projecte, no existia. L'equip d'*Angular* assegura que la migració de codi cap a la versió 2 no ha de ser problemàtica si s'ha estructurat el codi en components, cosa que hem fet en aquest projecte.

8.3 Aportació personal

Com a desenvolupador professional *full-stack* d'eines de gestió web, acostumat a usar metodologies tradicionals, aquest projecte m'ha donat el marc de treball que necessitava per a poder explorar el món de les *SPA* en profunditat. Efectivament, l'experiència d'usuari que s'obté és a anys llum de la que proporcionen les metodologies tradicionals, i el meu interès en vers a aquestes pràctiques només ha fet que créixer.

D'altra banda, aquest projecte també m'ha permès explorar el meu cantó obsessiu per la usabilitat i la comprensió de les interfícies d'usuari. Ha estat un repte, i també un procés molt satisfactori, analitzar i intentar millorar una eina prou complexa com l'*Authoring Tool* d'escriptori, i els fluxos de treball que aquesta permetia.

Finalment, aquest projecte ha ajudat a assentar els coneixements de desenvolupament de software obtinguts durant la carrera, obligant-me a seguir un meticulós procés d'anàlisi, disseny i implementació, amb models conceptuals, diagrames de classes i de seqüència, i patrons de disseny.

---

[17] *Angular 2* https://angular.io

Per últim, m'agradaria agrair la paciència de totes les persones del meu entorn, en un projecte que s'ha estès exageradament en el temps, i, sobretot, a la directora del projecte Carme Quer Bosor, que, a banda de pacient, ha estat sempre predisposada a prioritzar els meus interessos personals en el projecte sobre qualsevol altre.

Bibliografia

Fernàndez, P. M. (2010). *Desenvolupament de noves funcionalitats i millores de l'aplicació de gestió de qüestions de LEARN-SQL.* UPC.

Toporcer, K. A. (2007). *Gestor de cuestiones SQL y servicios web correctores.* UPC.

SPA - Single Page Applications (https://en.wikipedia.org/wiki/Single-page_application)

Front-Controller (https://en.wikipedia.org/wiki/Front_controller)

MVC - Model-View-Controller (https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller)

MVC - Model-View-Controller (http://www.moock.org/lectures/mvc/)

Apache Struts2 (https://struts.apache.org/)

AngularJS (https://angularjs.org/)

EmberJS (http://emberjs.com/)

ReactJS (https://facebook.github.io/react/)

GWT - Google Web Toolkit (http://www.gwtproject.org/)

Angular UI-Router (https://ui-router.github.io/)

Restangular (https://github.com/mgonto/restangular)

Bower (https://bower.io/)

Grunt (http://gruntjs.com/)

Flexbox (https://www.w3.org/TR/css-flexbox-1/)

JSON (http://www.json.org/)

Apache Struts JSON-Plugin (https://struts.apache.org/docs/json-plugin.html)

Apache Tomcat (http://tomcat.apache.org/)

UrlRewriteFilter (http://tuckey.org/urlrewrite/)